Université catholique de Louvain
Ecole polytechnique de Louvain
Laboratoire de Microélectronique

On graph-based cryptographic hash functions

Christophe Petit

Thèse soutenue en vue de l'obtention du grade de
Docteur en Sciences Appliquées

Composition du jury:

Pr Jean-Jacques Quisquater (UCL - DICE) – Promoteur
Pr Jean-Pierre Tignol (UCL - MATH) – Promoteur
Pr Olivier Pereira (UCL - DICE)
Dr Kristin Lauter (Microsoft Research)
Pr Bart Preneel (KULeuven)
Pr Gilles Zémor (Bordeaux I)
Pr Jean-Pierre Raskin (UCL - EMIC) – Président

Louvain-la-Neuve, Belgique
May 2009

Abstract

Hash functions are an invaluable tool for cryptography. They must primarily satisfy collision resistance, but standardized hash functions like SHA also satisfy stronger properties needed for the wide range of their applications. The design of many hash functions including SHA is based on a compression function that is close to a block cipher and on a domain extension transform like Merkle-Damgård. However, recent attacks against the collision resistance of SHA-1 suggest investigating new designs.

The expander hash design, proposed in the early nineties by Zémor and Tillich and recently rediscovered by Charles, Goren and Lauter, consists in defining a cryptographic hash function from an expander graph. The design is simple and elegant and important hash function properties can be interpreted as graph properties. When Cayley expander graphs are used, collision resistance reduces to the hardness of group-theoretical problems. Although these problems are not classical in cryptography, they appear in different forms in other fields and in at least one case, they have remained unbroken since 1994.

This thesis studies the expander hash design, its main strengths and weaknesses and the security and efficiency of currently existing instances. We introduce new functions, the Morgenstern hash function and the vectorial and projective versions of the Zémor-Tillich function. We study the security of particular constructions. We present new algorithms breaking the preimage resistance of the LPS hash function and the collision and preimage resistances of the Morgenstern hash function. We improve collision and preimage attacks against Zémor-Tillich and we describe hard and easy components of collision search for this function. We capture the malleability of expander hashes by two definitions of the literature and we describe its positive and negative consequences for applications. Finally, we introduce ZesT, an all-purpose hash function based on Zémor-Tillich, keeping its provable collision resistance and its parallelism but avoiding its malleability. Our function is provably secure, parallelizable, scalable, admits a wide range of (very) efficient implementations and can be used as a general-purpose hash function.

Acknowledgements

My first thanks for this thesis go to Professor Jean-Jacques Quisquater for accepting to serve as my advisor. Even though Jean-Jacques sometimes appears to be flying around the world, he has always also been a safe haven to me. I owe him these years at UCL CRYPTO group, very interesting collaborations with top international researchers, great scientific and career advices, interesting discussions and a careful review of my scientific work.

Professor Olivier Pereira has been perfectly complementary to Jean-Jacques. I appreciated his great availability for patiently explaining me subtleties of security definitions and proofs, his encyclopedic knowledge of the literature and his meticulous review of my thesis and some of my papers.

I am grateful to Dr. Kristin Lauter and to Professor Daniele Micciancio for arranging my stay in San Diego. These three months were a very great life experience both from personal and scientific points of view. I am especially grateful to Kristin for the collaboration we started, the work we published together, the lights she provided me on the mathematics subjacent to LPS and Pizer hashes, and finally for accepting to serve in my thesis committee. Many thanks also for the funny moments I spent with you and your family in San Diego.

I am also very grateful to Professor Gilles Zémor and to Dr. Jean-Pierre Tillich for the interesting and fruitful collaboration we started these last months. I owe them two of my favorite works so far, patient explanations of the ideas that lead them to invent the expander hash design and a careful review of large parts of this thesis. Thank you also, Gilles, for accepting to serve in my thesis committee.

I would also like to thank Professor Jean-Pierre Tignol for patiently teaching me on quaternion algebras and elliptic curves and for his very conscientious review of this text, and Professor Bart Preneel for accepting to serve in my thesis committee.

I thank all my co-authors and the anonymous reviewers of my papers. What I have learnt from each of you has served and will keep serving me in the future. I thank Nicolas and Giacomo whose help was precious in evaluating the performances of ZesT. I would also like to thank all the members of the UCL CRYPTO group and of the UCSD Cryptography and Security group for the atmospheres full of friendship and of research stimulation that I found in both places. I am especially grateful to Sylvie Baudine, heart and soul of the UCL Crypto group, for her permanent enthusiasm, her incredible efficiency and for her tracking down English errors throughout this text.

As I remember some periods of doubts and stress during these years, I also remember receiving strong support and advices from colleagues that I like to call my friends; I remember playing games and drinking beers with them and I remember our Tuesday's gastronomic lunches. With all these respects, I have special thanks to address to François, Julien, Olmar, Chong-Hee, Giacomo and Philippe.

I thank the Belgian National Science Foundation (FRS-FNRS) for its financial support and the excellent working conditions I benefited from during my thesis.

Finally, I would like to thank my family and all my friends for their friendship and support in life since I have known them. For everything, un tout grand merci, thank you so much, muchísimas gracias.

Contents

Chapter 1

Cryptographic hash functions from expander graphs

Cryptography has long been the art of spies and soldiers. Nowadays, it is used everyday by billions of people for securing electronic mail and payment transactions. In the last thirty years, it has also emerged as a science in its own right at the crossing point of mathematics and computer science.

The aim of cryptography is to protect information from being stolen or modified by malicious adversaries. This protection includes integrity, authenticity and confidentiality: information should not be modified without detection, documents and identities should be authenticated and no secret information should leak.

In modern cryptography, specific security goals are achieved either with specially designed algorithms or with the help of some mathematical problems. The first approach, inherited from block cipher designs, consists in using algorithms designed to achieve some kinds of random functions as well as possible. The second approach relies on number-theoretic and group-theoretic problems that are widely believed to be hard to solve.

This second approach is preferred throughout this thesis. From a theoretical point of view, it is more satisfactory as it captures the adversary's goal into a short, well-defined mathematical problem that can be studied by mathematicians independently of the protocol itself. However, with this approach cryptographic algorithms have a strong mathematic structure that can be used to attack them outside their original application. At the end of the thesis, we consequently mix both approaches in the design of the hash function ZesT.

Hash functions are a fundamental tool in cryptography. While their primary uses were digital signatures and message authentication codes, they appear nowadays in a wide range of applications requiring various properties. The most important properties ones are preimage and collision resistance: it must be computationally hard to invert a hash function or to find two inputs with the same output. Besides, hash functions are often used as perfectly random functions, in which case collision resistance does not suffice.

There exist old hash function constructions whose collision resistance follows from the hardness of number-theoretical and group-theoretical problems [120, 121, 82, 244, 71]. However, these functions can only be used in applications that only require collision resistance and they are also often too slow for practice. On the other hand, standardized hash algorithms like SHA follow the block cipher design: their use is therefore not restricted to collision resistance but their collision resistance is heuristic as it is not established by any concise and elegant mathematical problem. Actually, recent breakthroughs against the SHA-1 algorithm have questioned its design, which led NIST[1] to prompt a competition for a new Standard Hash Algorithm [265, 1].

The expander hash design goes back to 1991, when Zémor proposed to build a hash function from a Cayley graph of a special linear group [274]. This first construction was rapidly broken, but shortly later Tillich and Zémor proposed a second construction that was resistant to the previous attack [258] and remains essentially unbroken today. More than ten years later, Charles, Goren and Lauter [68] rediscovered the expander hash design and proposed the use of LPS and Pizer Ramanujan graphs.

The expander hash design fundamentally differs from classical hash designs and is very elegant. It allows relating important properties of hash functions like their collision resistance, their preimage resistance and their output distribution to the graph-theoretical notions of cycle, girth and expanding constants. When the graphs used are Cayley graphs, the design additionally provides efficient parallel computation and group-theoretical interpretations of the main hash properties.

The expander hash design, although more than 15 years old, is not very well known by the cryptographic community. The Zémor-Tillich hash function is sometimes considered as broken because of existing trapdoor attacks and attacks against particular parameters. The recent attacks against the LPS and Morgenstern hash functions have led many people to believe that the whole design is invalidated, while these attacks actually exploited particular structure of the LPS and Morgenstern graphs.

Indeed, many natural questions on expander hashes do not find answers in the literature. Relations between hash, graph and group properties were sketched in the papers but no precise statement of these relations exist. As

[1] American National Institute for Standards and Technology [4]

the mathematical problems underlying the security of expander hashes do not belong to classical problems, it is not clear whether the community actually tried to solve them. Hence their actual hardness remains to be established. Efficiency aspects have also only been sketched in the literature. Finally, expander hashes have been seen as theoretical hash functions (meaning with no chance of being once used in practice) due to their inherent malleability weaknesses originating from their mathematical structure.

The goal of this thesis is to formalize and prove general properties of expander hashes, to investigate the actual security and efficiency of existing constructions and to provide solutions for the inherent weaknesses of the design. The thesis covers all aspects of expander hashes, from applications of hash functions to security properties and from the security of particular instances to their practicability and their efficiency in software and in hardware implementations. In particular, we present new constructions and new attacks against existing constructions, we study the efficiency and practicability of all existing constructions and we propose approaches to solve the malleability issues; these achievements are detailed throughout the thesis and in Section 10.2.

The thesis is organized in three parts totalizing ten chapters, plus five appendices. Part I contains two chapters introducing cryptography and hash functions and describing the existing "provable" hash functions, which security relates to short mathematical problems. This part can be safely skipped by cryptographers.

Part II presents the expander hash construction and studies its security. Chapter 4 introduces the construction, formalizes security properties, presents existing instances and relates the design to other hash functions. Chapter 5 reviews cryptanalytic results on the Zémor-Tillich hash function, presents new attacks and introduces two variants of the function with reduced output sizes but the same security. Chapter 6 shows how to find collisions and preimages for the LPS and Morgenstern hash functions and Chapter 7 is dedicated to the Pizer hash function.

Part III opens perspectives and contains three chapters. Chapter 8 discusses malleability properties of hash functions and their consequences in applications, with a focus on the malleability properties of expander hash functions. Chapter 9 introduces the ZesT hash function, a provable hash function based on Zémor-Tillich that does not have the original weaknesses of the function but keeps its provable security and its parallelism. Finally, Chapter 10 concludes the thesis, summarizes known results of expander hashes, enlightens scientific contributions of the thesis and presents important open problems.

The appendices provide accessory and background information. Appendix A contains an exhaustive list of publications by the author; Appendix B gives some background and pointers to references in mathematics and computer science specific topics; Appendix C gives detailed hash algorithms for particular parameters of LPS and Morgenstern hash functions; Appendix E gives examples for our collision and preimage algorithms of Chapter 6.

We now provide background information on cryptographic hash functions. The reader willing to skip this general literature review may go directly to the description of expander hashes in Chapter 4. Those looking for specific information will find pointers to the relevant sections in the index or in the table of contents.

Part I

Introduction

Chapter 2

Cryptographic hash functions

Hash functions are one of the most useful but one of the less understood tools in modern cryptography. Despite of their common "hash function" denomination, the theoretical functions used to construct message authentication codes, digital signature schemes or pseudorandom number generators are very different. Depending on the definition, hash functions may be easy to construct but of little use, or like "random oracles", very useful in theory but practically impossible to build.

In all these definitions, hash functions are *compressing* functions, mapping *messages* of large, arbitrary size to *hash values* of small, constant size (Figure 2.1):

$$H : \{0,1\}^* \to \{0,1\}^\lambda.$$

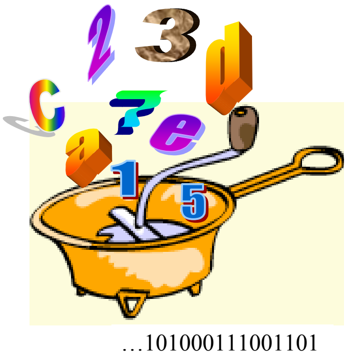

Figure 2.1: Representation of a hash function

The weakest notion of cryptographic hash functions, universal hash functions, only requires the output of the hash function to be well distributed.

The most popular ones, preimage and collision resistance, require that it is *computationally hard* to invert the function or to find two messages with the same hash values. The strongest notion, the random oracle model, considers the hash function as a "perfectly random function".

To formally capture the meaning of English words like "computationally hard", security definitions for hash functions actually apply to *families* of functions $\{H_n\}_{n\geq 0}$ parameterized by a *security parameter* n (rather than to a single function H), *i.e.*

$$H_n : \{0,1\}^* \rightarrow \{0,1\}^{\lambda(n)}$$

for some function $\lambda(n)$. In the literature, a hash function is actually defined as a family of *keyed* hash functions

$$H_n : \{0,1\}^{\kappa(n)} \times \{0,1\}^* \rightarrow \{0,1\}^{\lambda(n)}$$

for some functions $\kappa(n)$ and $\lambda(n)$. Although the keys might not always be mandatory to establish meaningful security proofs of protocols [231], they nevertheless seem necessary to formally define the notion of collision resistance in the standard computational model. Sometimes the message size is fixed in which case the hash function is called a *fixed-length* hash function, that is

$$H_n : \{0,1\}^{\kappa(n)} \times \{0,1\}^{\mu(n)} \rightarrow \{0,1\}^{\lambda(n)}$$

for some functions $\kappa(n)$, $\mu(n)$ and $\lambda(n)$.

Although the formal definitions apply to families of (keyed) hash functions, current standards and most alternative algorithms are unkeyed and have just been defined for a few values of the security parameter. It is current practice to replace the firsts by the seconds in concrete instantiations of protocols although this approach is flawed in general. Indeed, we may think of protocols that are secure if the key is randomly chosen but that behave in a particular insecure way on particular keys corresponding to these concrete instances. Since expander hashes may be defined asymptotically, we mostly work with formal definitions in this thesis. Unformal definitions are used along the text to help providing some intuitions, and in Chapter 9 when we turn the ZesT hash function into an unkeyed function fulfilling all NIST's requirements.

This chapter does not present original contribution but reviews the basics of cryptographic hash functions. Section 2.1 describes the computational security framework for cryptographic proofs. Sections 2.2 and 2.3 provide formal definitions of the most important notions of hash functions. Section 2.4

recalls the Merkle-Damgård construction, a building block in most cryptographic hash functions. Section 2.5 sketches the main general attacks against hash functions. Section 2.6 discusses applications and Section 2.7 concludes the chapter and provides pointers to the hash function literature.

2.1 Meaning of “security” in cryptography

2.1.1 Computational security

In this section, we give a formal meaning to the English words *efficient algorithm*, *negligible probability* or *computationally hard*. The definitions below are taken from [150]; we also follow the lines of their exposition but change their example from encryption algorithms to collision-resistant hash functions.

Let us consider an intuitive definition of a collision-resistant hash function: “it is hard to find a collision, that is a couple of messages (m, m') such that $H(m) = H(m')$”.[1] The exact meaning of “collisions are hard to find” is not “there exists no collision”. Indeed, as hash functions map large input sets onto small output sets, collisions always exist by the pigeon-hole principle. One may also try to formalize it as “there exists no algorithm able to produce a collision”, but let us consider the following two algorithms:

- **Algorithm** A_1 constructs a database of couples $(m_i, H(m_i))$. Until it finds a collision, A_1 picks a random message m_i, computes $h_i = H(m_i)$, checks the database for a previous occurrence of h_i (in which case it has found a collision). If h_i has not yet appeared it stores the new couple (m_i, h_i) in the database, otherwise it returns the collision found.

- **Algorithm** A_2 picks two random messages m and m'; it returns (m, m') if $H(m) = H(m')$ and returns $\perp$ otherwise.

Algorithm A_1 always finds collisions after at most $2^{\lambda} + 1$ hash computations (and about $2^{\lambda/2}$ in mean, see Section 2.5), and Algorithm A_2 produces collisions with a probability at least $1/2^{\lambda}$. However, for λ large enough (in practice λ is at least 128), any existing computer would require many years and a prohibitively huge memory to execute Algorithm A_1, while Algorithm A_2 will succeed with a probability so small that it can be neglected in practice.

[1] As we have already mentioned, the formal definition for collision-resistant hash functions actually requires families of functions rather than functions. We set this technicality apart until the next section.

As unconditional security cannot be achieved, cryptographic definitions take place in the weaker framework of *computational security.* The algorithms are required to be *efficient* and adversarial algorithms may succeed with some with some *negligible* probability.

There are two main approaches for computational security. In the *concrete* approach, the security definitions are parameterized by concrete numbers: some protocol will be (ϵ, t, m)-secure in some sense if any algorithm running in time less than t and using a memory smaller than m, succeeds in some task with probability smaller than ϵ. This approach fits well to practice, for example with $t = 2^{60}$, $m = 2^{40}$ and $\epsilon = 2^{-40}$.

The second approach is the *asymptotical* approach. In the case of hash functions, the asymptotical definitions apply to a family of hash functions $\{H_n\}$ indexed by a *security parameter* n rather than to one single function H. The hash family $\{H_n\}$ is secure if each H_n is $(\epsilon(n), t(n), m(n))$-secure, the functions $t(n)$ and $m(n)$ do not grow too fast with n, and the function $\epsilon(n)$ decreases fast enough with n.

Definition 2.1 *An algorithm A is* efficient *or* probabilistic polynomial time *or* PPT *if there exists a polynomial $p(.)$ such that for every input $x \in \{0,1\}^*$, the computation of $A(x)$ (that may involve some probabilistic choices) terminates within at most $p(|x|)$ steps. (For $x \in \{0,1\}^*$, $|x|$ denotes the length of the string x.)*

Definition 2.2 *A function f is* negligible *if for every polynomial $p(.)$ there exists an N such that for all integers $n \geq N$ it holds that $f(n) < \frac{1}{p(n)}$.*

Definition 2.3 *A function f is* noticeable *if there exists a polynomial $p(.)$ such that for all sufficiently large integers n, it holds that $f(n) > \frac{1}{p(n)}$.*

Although it does not explicitly appear in its definition, a PPT algorithm can not use more than a polynomial amount of memory: indeed, the time needed even only to read a super-polynomial amount of memory would be super-polynomial. The notions of negligible and noticeable functions are "strong negations" one to each other. In particular, there exist functions that are neither negligible nor noticeable.

An asymptotic security definition always looks as follows:

> The cryptographic scheme X is secure in the sense Y if for any PPT algorithm given an input of size n, there exists a negligible function $\epsilon(n)$ such that the algorithm succeeds with probability smaller than $\epsilon(n)$ in performing some task Z.

Asymptotic definitions are well suited for theory but they do not often fit with hash functions practice, as standard algorithms like SHA are only defined for a few values of the security parameter.

Concrete and asymptotic approaches are complementary. In most of the thesis we use the asymptotic formalism because it allows for "cleaner" proofs and unlike most hash algorithms, the security of expander hashes can be expressed in this formalism. In Chapters 5 and 9, we also use the concrete approach to evaluate the actual security of the Zémor-Tillich and ZesT hash functions with respect to the best known attacks.

2.1.2 Security reductions

Many theorems in cryptography have the following form:

> If the computational assumption X holds, then the cryptographic scheme Y is secure in the computational sense Z.

Such a theorem *reduces* the confidence one can have on the security of X to the confidence one has that the assumption X is true. If X turns out to be false, this theorem will say nothing about the security of the algorithm Y. Fortunately, there exist widely believed computational assumptions based on number theoretic problems that have been challenging mathematicians and cryptographers for decades or even centuries.

The integer factorization assumption, the discrete logarithm assumption and the elliptic curve discrete logarithm assumption state that the following problems are computationally hard to solve.

- **Integer Factorization Problem:** given a large composite number of the form $n = pq$ for p, q primes, compute p and q.

- **Discrete Logarithm Problem:** given a prime p, an element g of $\mathbb{F}_p$ with large prime order, and the element $g^k \bmod p$ for some randomly chosen k, return the k value.

- **Elliptic Curve Discrete Logarithm Problem:** given an elliptic curve E defined over a prime field $\mathbb{F}_p$, a rational point $P \in E$ with some large order, and the point $Q = kP$ for some randomly chosen k, return the k value.

A security proof *by reduction* is a proof *ad absurdum*. The proof first supposes that there exists an algorithm A breaking the cryptographic scheme Y with a non-negligible probability. It then describes a *reduction algorithm*

A' that uses A as a subroutine to solve instances of the cryptographic scheme Z. Finally, the proof shows that A' contradicts the assumption X, that is A' solves the corresponding problem with a non-negligible probability.

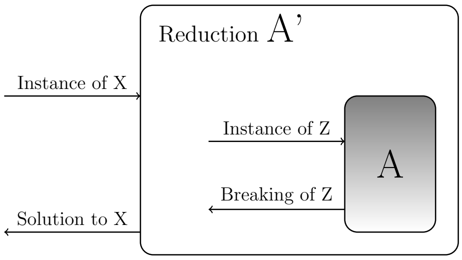

Figure 2.2: Security proof by reduction

This proof technique has been widely applied in cryptography: the existence of most cryptographic tools has been reduced to the existence of only two primitives, trapdoor permutations and one-way functions, and both of them can be constructed under a few widely-believed number theoretical computational assumptions [136, 226].

2.2 Preimage, second preimage and collision resistances

Preimage, second preimage and collision resistances are certainly the most popular security requirements for hash functions. Their intuitive meanings, as given by [176] are the following:

- **Preimage resistance:** for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, *i.e.*, to find any preimage m' such that $H(m') = h$ when given any h for which a corresponding input is not known.

- **Second preimage resistance:** it is computationally infeasible to find any second input which has the same output as any specified input, *i.e.*, given m, to find a second preimage $m' \neq m$ such that $H(m) = H(m')$.

- **Collision resistance:** it is computationally infeasible to find any two distinct inputs m, m' which hash to the same output, *i.e.*, such that $H(m) = H(m')$.

We have already given a formal meaning to "computationally infeasible". Rogaway and Shrimpton pointed out that the words "essentially all" may also induce their lot of ambiguities [232]. For families of unkeyed hash functions there is no sense in saying that "it must be computationally hard to compute a preimage of 0", because a (non-uniform) polynomial-time algorithm may very well have stored in some database one preimage of 0 for each value of the security parameter and simply access to and return it. For the definition to make sense, the challenge that the attacker is asked to solve should not be known in advance. Formal definitions of preimage resistance consequently include some randomness, either in the value h to which the adversary must find a preimage, or in the function itself in the case of keyed functions.

In this section, we give asymptotic versions of the seven definitions of preimage, second preimage and collision resistant hash functions proposed by Rogaway and Shrimpton, as well as the implications they found out between the different notions [232]. The definitions apply to keyed hash functions, but the notions `aPre` and `aSec` are also meaningful for unkeyed hash functions because they fix the key.

First of all, we define a hash function without any particular cryptographic strength.

Definition 2.4 *[150] A* hash function *is a pair of* PPT *algorithms* (Gen, H) *such that*

- ***Polynomial-time indexing:*** *Gen takes as input* 1^n *(a string of* n *"ones", where* n *is the security parameter) and outputs a key* s *(which implicitly contains* 1^n*).*
- ***Polynomial-time evaluation:*** *There exists a polynomial* λ *such that* H *takes as input a key* s *and a string* $m \in \{0,1\}^*$ *and outputs a string* $H(s,m) \in \{0,1\}^{\lambda(n)}$*.*

We say that (H, Gen) is a *fixed-length hash function* if the message length is fixed for each value of the security parameter, that is $m \in \{0,1\}^{\mu(n)}$ for some function $\mu(n)$.

In this definition, the security parameter n is implicit in the key s and the algorithm H implicitly defines a family of keyed functions $H_n : \{0,1\}^{\kappa(n)} \times \{0,1\}^* \to \{0,1\}^{\lambda(n)}$. We point out that unlike in other cryptographic contexts, the keys of hash functions are not aimed to be secret values but rather to provide a source of randomness necessary in the definitions. All the security notions of this section hold in the following definition.

Definition 2.5 - *A hash function (Gen, H) is* `ePre`*-secure (resp.* `Col`*-secure) if for any* `PPT` *algorithm A, the probability $Adv_{(Gen,H)}^{\texttt{ePre},A}$ (resp. $Adv_{(Gen,H)}^{\texttt{Col},A}$) that A wins some game related to* `ePre` *(resp.* `Col`*) is negligible, where the corresponding probabilities $Adv_{(Gen,H)}^{\texttt{ePre},A}$ and $Adv_{(Gen,H)}^{\texttt{Col},A}$ are as defined in the following subsections.*

- *A hash function (Gen, H) is* `NOTION`*-secure for messages of length μ if for any* `PPT` *algorithm A, the probability $Adv_{(Gen,H)}^{\texttt{NOTION}[\mu],A}$ that A wins some game related to* `NOTION` *and parameterized by a function μ, is negligible. A hash function (Gen, H) is* `NOTION`*-secure if for any noticeable function ϵ, it is* `NOTION`*-secure for messages of length μ where $\mu(n) = (1 + \epsilon(n))\lambda(n)$ and λ is given by Definition 2.4. Here* `NOTION` *can be any of* `Pre`*,* `aPre`*,* `Sec`*,* `eSec`*,* `aSec` *and the corresponding probabilities $Adv_{(Gen,H)}^{\texttt{NOTION}[\mu],A}$ are as defined in the following subsections.*

The second part of this definition may appear a bit tricky at first sight, but bounding the message length is necessary for the implication between collision and preimage resistance [232] to hold (see Section 2.2.4).

2.2.1 Preimage resistance

Rogaway and Shrimpton distinguish three notions of preimage resistance, depending on where the randomness is introduced. For the notion `aPre`, the challenge is random but the key is fixed: the function must be "always" preimage resistant, that is for any fixed key. For `ePre`, the key is random but the challenge is fixed: the function is "everywhere" preimage resistant, that is for any fixed challenge. Finally, for the notion `Pre` both the challenge and the key are random. This latest notion has also been called *one-way function.*

Always Preimage Resistance:

$$Adv_{(Gen,H)}^{\texttt{aPre}[\mu],A}(n) = \max_{s \in Gen(1^n)} \Pr\left[Exp_{(Gen,H)}^{\texttt{aPre}[\mu],s,A}(n) = 1\right],$$

where $Exp_{(Gen,H)}^{\texttt{aPre}[\mu],s,A}(n)$ is the $Exp_{(Gen,H)}^{\texttt{aPre}[\mu],s,A}(n)$ is the following experiment

Experiment $Exp^{\mathtt{aPre}[\mu],s,A}_{(Gen,H)}(n)$:

- a random message m is picked uniformly in $\{0,1\}^{\mu(n)}$;
- the hash value $h = H(s,m)$ is given to A;
- A outputs some message $m' \in \{0,1\}^*$;
- $Exp^{\mathtt{aPre}[\mu],s,A}_{(Gen,H)}(n) = 1$ if and only if $H(s,m') = h$.

Everywhere Preimage resistance:

$$Adv^{\mathtt{ePre},A}_{(Gen,H)}(n) = \max_{h \in \{0,1\}^{\lambda(n)}} \Pr\left[Exp^{\mathtt{ePre},h,A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\mathtt{ePre},h,A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\mathtt{ePre},h,A}_{(Gen,H)}(n)$:

- a key s is generated by running Gen on input 1^n;
- the key s is given to A;
- A outputs some message $m \in \{0,1\}^*$;
- $Exp^{\mathtt{ePre},h,A}_{(Gen,H)}(n) = 1$ if and only if $H(s,m) = h$.

Preimage Resistance:

$$Adv^{\mathtt{Pre}[\mu],A}_{(Gen,H)}(n) = \Pr\left[Exp^{\mathtt{Pre}[\mu],A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\mathtt{Pre}[\mu],A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\mathtt{Pre}[\mu],h,A}_{(Gen,H)}(n)$:

- a key s is generated by running Gen on input 1^n
- a random message m is picked uniformly in $\{0,1\}^{\mu(n)}$;
- the key s and the hash value $h = H(s,m)$ are given to A;
- A outputs some message $m' \in \{0,1\}^*$;
- $Exp^{\mathtt{Pre}[\mu],A}_{(Gen,H)}(n) = 1$ if and only if $H(s,m') = h$.

2.2.2 Second preimage resistance

As for preimage-resistance, Rogaway and Shrimpton introduce three notions of second preimage resistance: `aSec` or "always" second preimage resistance, `eSec` or "everywhere" second preimage resistance, and `Sec` or second preimage resistance. The second notion is equivalent to the notion of *universal one-way hash function family* (UOWHF) introduced by Naor and Yung [192]. The last one is sometimes called *weak collision resistance*.

Always Second Preimage Resistance:

$$Adv^{\mathtt{aSec}[\mu],A}_{(Gen,H)}(n) = \max_{s \in Gen(1^n)} \Pr\left[Exp^{\mathtt{aSec}[\mu],s,A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\mathtt{aSec}[\mu],s,A}_{(Gen,H)}(n)$ is the $Exp^{\mathtt{aSec}[\mu],s,A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\mathtt{aSec}[\mu],s,A}_{(Gen,H)}(n)$:

- a random message m is picked uniformly in $\{0,1\}^{\mu(n)}$;
- the message m is given to A;
- A outputs some message $m' \in \{0,1\}^*$;
- $Exp^{\mathtt{aPre}[\mu],s,A}_{(Gen,H)}(n) = 1$ if and only if $m \neq m'$ and $H(s,m') = H(s,m)$.

Everywhere Second Preimage resistance:

$$Adv^{\mathsf{eSec}[\mu],A}_{(Gen,H)}(n) = \max_{m \in \{0,1\}^{\mu(n)}} \Pr\left[Exp^{\mathsf{eSec},m,A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\mathsf{eSec},m,A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\mathsf{eSec},m,A}_{(Gen,H)}(n)$:

- a key s is generated by running Gen on input 1^n;
- the key s is given to A;
- A outputs some message $m' \in \{0,1\}^*$;
- $Exp^{\mathsf{eSec},h,A}_{(Gen,H)}(n) = 1$ if and only if $m \neq m'$ and $H(s,m) = h$.

Second Preimage Resistance:

$$Adv^{\mathsf{Sec}[\mu],A}_{(Gen,H)}(n) = \Pr\left[Exp^{\mathsf{Sec}[\mu],A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\mathsf{Sec}[\mu],A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\mathsf{Sec}[\mu],h,A}_{(Gen,H)}(n)$:

- a key s is generated by running Gen on input 1^n
- a random message m is picked uniformly in $\{0,1\}^{\mu(n)}$;
- the key s and the message m are given to A;
- A outputs some message $m' \in \{0,1\}^*$;
- $Exp^{\mathsf{Sec}[\mu],A}_{(Gen,H)}(n) = 1$ if and only if $H(s,m') = H(s,m)$.

2.2.3 Collision resistance

In the collision experiment there is no challenge, hence there is only one definition of collision resistance. This notion is also sometimes called *strong collision resistance* or *collision-freeness.*

Collision Resistance:

$$Adv^{\texttt{Col},A}_{(Gen,H)}(n) = \Pr\left[Exp^{\texttt{Col},A}_{(Gen,H)}(n) = 1\right],$$

where $Exp^{\texttt{Col},A}_{(Gen,H)}(n)$ is the following experiment

Experiment $Exp^{\texttt{Col},A}_{(Gen,H)}(n)$:

- a key s is generated by running Gen on input 1^n;
- the key s is given to A;
- A outputs two messages $m, m' \in \{0,1\}^*$;
- $Exp^{\texttt{Col},A}_{(Gen,H)}(n) = 1$ if and only if $H(s,m') = H(s,m)$.

2.2.4 Implications

Roughly, collision resistance implies preimage and second preimage resistance, but the converse is not true [249]. More nuances appear for fixed-length hash functions; in Figure 2.3, we give the implications between the different notions as they are provided in [232]. An arrow from `notion1` to `notion2` means that any hash function that is secure in the sense of `notion1` is also secure in the sense of `notion2`.

The dashed arrows of Figure 2.3 only hold when the message set is large enough. In our Definition 2.4 of a hash function, the input of H is a string of arbitrary length. In the case of fixed-length hash functions, when the input is restricted to a set of size 2^{μ}, the dashed arrows of Figure 2.3 may not be true anymore. For example, the inequality

$$Adv^{\texttt{Pre}[\mu],A}_{(Gen,H)}(n) \leq Adv^{\texttt{Sec}[\mu],A}_{(Gen,H)}(n) + 2^{\lambda(n)-\mu(n)}$$

from [232] is represented by the dashed arrow between `Sec` and `Pre`. It is meaningful only if the ratio between the input and output sets is large enough. We refer to [232] for further details on the inequalities represented by these arrows.

2.3 Other security notions for hash functions

Preimage, second preimage and collision resistance have become the most popular security notions for hash functions, but there exist many other definitions.

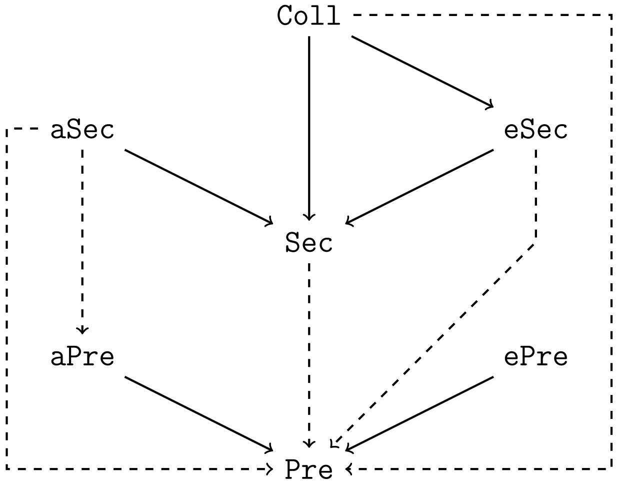

Figure 2.3: Relationships among the different security notions of preimage, second preimage and collision resistance [232]

Hash functions have numerous applications in cryptographic protocols. As preimage and collision resistance are not always necessary for the security of these protocols, researchers have been looking for weaker notions that provide the same security guaranties under slighter assumptions. On the other hand, some very standard protocols using a hash function are insecure in general if the function does not satisfy some security requirement beyond collision resistance; for these protocols there is a need of alternative, stronger security notions. Today, while the NIST competitors have been trying to *build* good hash functions, other researchers in the world are actually trying to *define* what kind of function should be built.

In this section, we define Universal Hash Functions (UHF), Pseudorandom Functions (PRF), Perfectly One-Way Hash Functions (POWHF) and we present the Random Oracle Model (RO), a very useful abstraction for cryptographic proofs.

2.3.1 Universal hash functions

Universal Hash Functions have been first introduced by Carter and Wegman [65]. Intuitively, the hash function used with a random key is required to behave like a random function on any input. The requirement for UHF is not computational: it is a statistical property that must hold even for non-PPT adversaries.

Definition 2.6 *A hash function (Gen, H) is a ϵ-almost Universal Hash Function (UHF) if for all n, for all $m, m' \in \{0,1\}^n$,*

$$\Pr_{s \in Gen(1^n)} [H(s,m) = H(s,m')] \leq \epsilon(n).$$

The case $\epsilon(n) = 2^{-\lambda(n)}$ (where $\lambda(n)$ is given by Definition 2.4) is called *universal* [65].

Definition 2.7 *A hash function (Gen, H) is a ϵ-almost Strongly Universal Hash Function (SUHF) if*

- *for all n, for all $m \in \{0,1\}^n$, for all $h \in \{0,1\}^{\lambda(n)}$,*

$$\Pr_{s \in Gen(1^n)} [H(s,m) = h] = 2^{-\lambda(n)};$$

- *for all n, for all $m, m' \in \{0,1\}^n$, for all $h, h' \in \{0,1\}^{\lambda(n)}$,*

$$\Pr_{s \in Gen(1^n)} [H(s,m) = h \wedge H(s,m') = h'] \leq \epsilon(n) 2^{-\lambda(n)},$$

where $\lambda(n)$ is given by Definition 2.4.

The smallest possible values for ϵ are given by $\epsilon(n) = 2^{-\lambda(n)}$, in which case the first condition follows from the second one. These functions are called *strongly universal* [268, 193].

Universal hash functions have been generalized in many ways. There exist k-wise versions of these definitions where the hash function with a random key is required to behave like a random function on any set of k inputs. The colliding condition $H(s,m) = H(s,m')$ has been generalized to $H(s,m) \oplus H(s,m') = 0$ and other relations in Abelian groups. We refer to [65, 155, 230, 255, 257, 254, 268, 193] for more details and other definitions.

2.3.2 Pseudo-random functions

A hash function is a Pseudorandom Function (PRF) if no PPT algorithm that is not given the key can distinguish its behavior from the behavior of a truly random function.

Formally, a truly random function is a function randomly chosen among some set of functions. For the definition to make sense, this set must be finite, so a random function with given domain and codomain is defined as a random function chosen uniformly among the set of all functions with the same domain and codomain.

In the definition of a PRF, the PPT adversary has a black-box access to some functionality called an *oracle* that implements either the random function or the PRF with a random key (also called *seed* in this context). A hash function is a PRF if no adversary can tell with a probability significantly better than 1/2 wether he is interacting with the PRF or the truly random function.

Definition 2.8 *A Hash Function* (Gen, H) *is a Pseudorandom Function (PRF) if for any* PPT *algorithm* A*, the function*

$$Adv^{PRF,A}_{(Gen,H)}(n) := \left|\Pr\left[Exp^{PRF,1,A}_{(Gen,H)}(n) = 1\right] - \Pr\left[Exp^{PRF,0,A}_{(Gen,H)}(n) = 1\right]\right|$$

is a negligible function, where $Exp^{PRF,1,A}_{(Gen,H)}(n)$ *and* $Exp^{PRF,0,A}_{(Gen,H)}(n)$ *are defined below.*

Experiment $Exp^{\mathrm{PRF},0,A}_{(Gen,H)}(n)$:	Experiment $Exp^{\mathrm{PRF},1,A}_{(Gen,H)}(n)$:
- a function f is chosen randomly among the set of functions with domain $\{0,1\}^n$ and codomain $\{0,1\}^{\lambda(n)}$;	- a key s is generated by running Gen on input n;
- A makes any number of *queries* to its oracle, that are messages $m_i \in \{0,1\}^n$;	- A makes any number of *queries* to its oracle, that are messages $m_i \in \{0,1\}^n$;
- the oracle answers to each query m_i by $f(m_i)$;	- the oracle answers to each query m_i by $H(s, m_i)$;
- A returns some bit b.	- A returns some bit b.

The requirement of being pseudorandom seems much stronger than collision resistance but the two notions cannot be related in general [229]. Unlike for collision resistance, the security of a hash function as a PRF entirely depends on the secrecy of the seed: the definition gives no security guaranty if the seed is known to the adversary. In particular, in this security model, the adversary must even be denied the ability to compute the hash function itself because this computation would require the knowledge of the seed.

2.3.3 Perfectly one-way probabilistic hash functions

Perfectly one-way functions (POWHF) were introduced in 1997 by Canetti [61] as part of a research program investigating new security definitions for hash functions to replace the random oracle model. The primitive was first called Oracle Hashing, and recalled POWHF by Canetti et al. [64]. POWHFs are aimed to be hash functions whose outputs hide all partial information on the corresponding inputs.

Perfectly one-way hash functions are probabilistic; unlike the functions of Definition 2.4, the output of the hash algorithm is not constant. To check correctness of a hash computation, probabilistic hash functions are given a third algorithm Ver.

Definition 2.9 *A* probabilistic hash function *is a triple of* PPT *algorithms* (Gen, H, Ver) *such that*

- ***Polynomial-time indexing:*** *Gen takes as input* 1^n *(a string of* n *"ones", where* n *is the security parameter) and outputs a key* s *(which implicitly contains* 1^n*).*
- ***Polynomial-time evaluation:*** *There exist polynomials* $\lambda(n)$ *and* $\rho(n)$ *such that* H *takes as input a key* s*, a string* $m \in \{0,1\}^*$ *and a* randomizer $r \in \{0,1\}^{\rho(n)}$*, and outputs a string* $H(s,m,r) \in \{0,1\}^{\lambda(n)}$*.*
- ***Completeness:*** *For any key* s*, message* m *and randomizer* r*, the probability* $\Pr[Ver(s,m,H(s,m,r)) \neq 1]$ *is a negligible function of* n*.*

A perfectly one-way hash function is a probabilistic hash function that is collision resistant and hides any partial information on its input.

Definition 2.10 *A perfectly one-way hash function (POWHF) is a probabilistic hash function* (Gen, H, Ver) *satisfying the following requirements.*

- ***Correctness/ Collision resistance:*** *for any* PPT *algorithm* A*, on input* s*, a triplet* (m, m', h) *such that* $m \neq m'$ *and* $Ver(s,m,h) = Ver(s,m',h) = 1$ *is negligible.*
- ***Secrecy:*** *For any* $s \in \{0,1\}^{\kappa(n)}$*, any* PPT *algorithm* A *with binary output and any* well-spread *distribution ensemble* $\{\mathcal{D}_n\}$*, the distributions of* $m||A(H(s,m,r)$ *and of* $m||A(H(s,m',r))$ *are* computationally indistinguishable *when* r *is randomly picked in* $\{0,1\}^{\rho(n)}$ *and* m, m' *are independently drawn according to* $\mathcal{D}_n$*.*

In this definition, a distribution ensemble $\{\mathcal{D}_n\}$ is *well-spread* if the largest probability of any element drawn by $\mathcal{D}_n$ is a negligible function of n. Two distribution ensembles are *computationally indistinguishable* if for any PPT algorithm A, the difference $|\Pr[A(x) = 1] - \Pr[A(y) = 1]|$ is a negligible function of n when x and y are drawn independently according to each of the two distribution ensembles.

The meaning of Definition 2.10 is that POWHFs hide all information on their inputs. An equivalent definition is provided by Canetti [61] meaning that the information received with the hash value of a message m is not more useful than the information provided by an oracle returning 1 on input m and 0 on any other input.

POWHFs have practical limitations but are an interesting step toward replacing random oracles. The security of POWHFs requires producing good trusted random bits for the randomizers, which is a non-trivial task in its own right. Moreover, many cryptographic protocols including standard ones have been built upon non-probabilistic hash functions. On the other hand, the notion captures important properties of random oracles and unlike random oracles, it can be instantiated based on standard cryptographic assumptions [61, 64, 63].

2.3.4 The Random Oracle model

The random oracle model has been proposed by Bellare and Rogaway [41], building on previous works that were implicitly using the same approach [117, 116, 100, 139, 162]. Random oracles have been widely used in cryptography to prove the security of simple and efficient protocols that could not be proved in the standard model (as opposed to the random oracle model). However, since the random oracle methodology is not sound [63], the current tendency in cryptography is to prefer protocols secure in the standard model even if they are less less efficient.

A random oracle receives as input messages $m_i \in \{0,1\}^*$ and returns hash values $h_i \in \{0,1\}^\lambda$ as follows: If it was not previously queried on m_i the oracle picks a random value $h_i \in \{0,1\}^\lambda$, stores (m_i, h_i) in some database that it maintains, and returns h_i. If it was previously queried on m_i it returns the corresponding h_i value stored in its database.

The random oracle paradigm, as described in [41] is the following: To devise a protocol P for some protocol problem Π,

1. Find a formal definition for Π in the model of computation in which all parties (including the adversary) share a random oracle R;

2. Devise an efficient protocol for Π in the random oracle model;

3. Prove that P satisfies the definition of Π;

4. Replace (*instantiate*) oracle access to R by computation of a "good" hash function.

By a "good" hash function, Bellare and Rogaway mean one with no apparent structure. In their paper, they propose simple modifications of MD5 [227] and SHA [14] that could be used for this purpose.

The random oracle paradigm has been and remains widely used in cryptography but, as pointed out by Canetti, Goldreich and Halevi [63] it does not provide any security guaranty by itself. Indeed, any signature or encryption schemes secure in the random oracle model can be transformed into other signature or encryption scheme that are still secure in the random oracle model but insecure for any concrete instantiation of the random oracle.

Similarly, Goldwasser and Kalai [119] have shown that the Fiat-Shamir heuristic (see Section 2.6.2), although secure in the random oracle model, is not sound either: there exists a three round authentication scheme, which does not give a secure signature scheme when Fiat-Shamir is applied to it with any efficient hash function.

Nielsen [194] has shown that non-interactive non-committing encryption schemes do exist in the random oracle model but not in the standard model. This result is stronger than the previous ones because it rules out any transformation from the random oracle model to the standard model, not only the transformations where the random oracles are instantiated by concrete functions.

To provide some intuition on these results, let us consider a weaker version of Canetti, Goldwasser and Halevi's result for signatures [63]: for any hash function $\mathcal{H} = (Gen, H)$, there exists a signature scheme secure in the random oracle model but insecure if instantiated with $\mathcal{H}$.

Suppose you have a signature scheme secure (see Section 2.6.2 for definitions) in the random oracle model. The modified signing algorithm is defined such that it remains unchanged in most inputs, but leaks the signature scheme's secret key on inputs m such that the oracle answer $R(m)$ belongs to the set $\{H(s,s)|s \in \{0,1\}^{\kappa(n)}\}$.

If this modified algorithm is instantiated with $\mathcal{H}$, after receiving the hash key s an adversary can simply ask for a signature on s, after which he gets the private key of the signature scheme and can sign any message of his choice. However, in the random oracle model the adversary is unlikely to find a message that will produce the key leakage, hence the scheme remains "secure".

This example and other ones leading to similar results may appear contrived, but they still provide a warning that a proof of security in the random oracle model does not give any guaranty in itself for the security of the scheme instantiated with a concrete hash function. No "natural" cryptographic scheme secure in the random oracle has ever been attacked when instantiated with a "good" hash function, and nobody will ever design a signature scheme that explicitly returns its secret key even on particular values. However, as these examples show flaws in the random oracle model, we may fear other flaws, potentially damaging, on more natural protocols. At least, these examples enlighten that our understanding of hash functions is still uncomplete and that better definitions are desirable.

Random oracles are practical but dangerous. Today, the cryptographic community is somehow divided into those who reject the above examples as unnatural and still use the random oracle model for its great practical advantages (although they acknowledge that the random oracle model does not strictly give them any security guaranty), and those who fear more damageable misunderstandings of random oracles and press people to avoid them (although they reckon that most applications of random oracles are probably safe). A proof in the random oracle model provides a useful sanity check on the protocol, but "natural" cryptographic schemes that are only proved secure in the random oracle model are only probably, not provably, secure in practice.

2.4 The Merkle-Damgård transform

Most cryptographic hash functions are built upon two main components: a *compression function* and a *domain-extension transform.* The compression function hashes messages of a fixed size to hash values of fixed, smaller size; the domain-extension transform uses the compression function as a building block to construct hash functions with arbitrary-length inputs.

The Merkle-Damgård transform (Figure 2.4) was independently discovered by Damgård [83] and Merkle [178]; it is used most notably in the MD_+ and SHA families of hash functions. The compression function f takes as input a key s and a message of size $\mu + \lambda$, and returns a bitstring of size λ; it is assumed to be collision-resistant. The Merkle-Damgård transform of f takes as input a key s and a message m of length L smaller[2] than 2^λ, and

[2] The restriction on the size of the messages has no practical influence. For parameters λ of interest, 2^λ is far larger than the size of messages that will be hashed in practice. Moreover, the MD transform can also process larger messages if the message lengths are coded on more than one block.

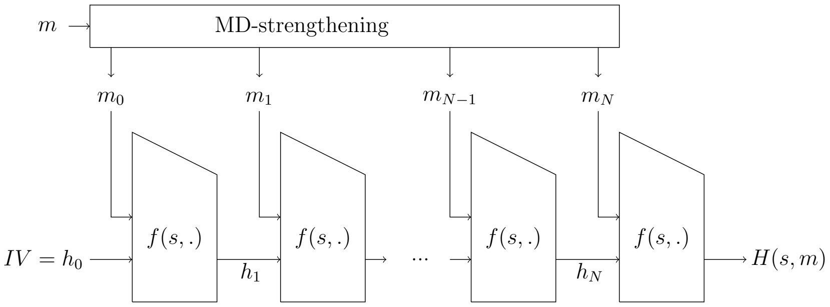

Figure 2.4: Merkle-Damgård transform

returns a bitstring of size λ.

From a message m, the *MD-strengthening* produces $N + 1$ bitstrings $m_0, ..., m_{N+1}$ of size μ, where $N = \lceil L/\mu \rceil$. The message m is first decomposed into N blocks of μ consecutive bits. If L is not a multiple of μ, the last block is completed with zeroes. An additional block is constructed that contains a binary representation of L on λ bits.

Let $h_0 = IV$ be some fixed *initial value.* The Merkle-Damgård transform of f is defined as $H_f(s,m) = h_{N+1}$, where

$$h_i = f(s, h_{i-1}||m_{i-1}).$$

The Merkle-Damgård transform satisfies the following property:

Theorem 2.1 (Merkle-Damgård) *If (Gen, f) is a fixed-length collision resistant hash function, then (Gen, H) is a collision-resistant hash function.*

The proof of this theorem is easy. Intuitively, suppose an adversary finds $m \neq m'$ such that $H_f(s,m) = H_f(s,m')$. Write m_i and m'_i for the output blocks of the MD-strengthening of m and m', and h_i and h'_i for the intermediate values of the computation of $H_f(s,m)$ and $H_f(s,m')$. Then there exists $i \leq n+1$ such that $h_i = h'_i$ but $h_{i-1}||m_{i-1} \neq h'_{i-1}||m'_{i-1}$, so the adversary has found a collision $(h_{i-1}||m_{i-1}, h'_{i-1}||m'_{i-1})$ on the compression on the compression function.

The assumptions on f in Theorem 2.1 can be slightly relaxed [76]. Damgård also proposed a variant of the transform, using a binary tree, that satisfies the same property and allows for bigger parallelism [83]. As computers in the nineties were unable to exploit this parallelism, the serial version was preferred for its shorter memory requirements.

The Merkle-Damgård transform is a domain-extending collision resistance-preserving transform: it transforms a fixed-length collision resistant hash function into an arbitrary-length collision resistant hash function. However, it does not preserve other properties like preimage resistance, second preimage resistance, pseudo-randomness and indistinguishability from a random oracle.

There exist many alternative transforms that preserve different properties. Shoup gave a transform preserving the UOWHF or everywhere preimage resistance property [246], but at the cost of a key length increase. The ROX transform of Andreeva et al. [28] preserves the seven security notions of Rogaway and Shrimpton. Coron et al. [78] gave a transform preserving the property of indistinguishability from a random oracle. Finally, the EMD transform of Bellare and Ristenpart [40] preserves collision resistance, pseudorandomness and indistinguishability from a random oracle.

In a different vein, Herzberg [130], Fischlin and Lehmann [103, 104] and Fischlin et al. [105] proposed multi-property combiners, combining various hash functions in such a way that if any of them satisfies one security property, the combined function also satisfies this property.

2.5 Popular attacks on hash functions

An attack on a cryptographic protocol is a proof that this protocol does not satisfy its claimed security properties. Attacks on hash functions have been mainly targeting the collision and preimage resistance properties.

In the asymptotic setting, an attack against the collision resistance of a hash function is a PPT algorithm that finds collisions for asymptotically large values of the security parameter. In practice, many hash functions are only defined for a finite small set of values of the security parameter, so the meaning of PPT algorithm lacks sense.

Because the codomain of any concrete hash function is a finite set, there exist attacks called *generic* that cannot be avoided but by fixing large enough parameters. The *exhaustive search attack* and the *random trial attack* find preimages for any hash function in a time proportional to the codomain size while the *birthday attack* only requires the square root of this time to find collisions. In practice, a preimage or a collision attack is often considered successful if it computes preimage or collisions faster than these generic attacks.

Iterated hash functions are not generic hash functions. Their inherent structure allows for more attacks, among which the *fixed point attack*, Joux' *multicollision attack*, and in some cases the *meet-in-the-middle attack* [145, 220]. These functions are also good targets for *differential cryptanalysis* because their compression functions share many characteristics with block ciphers [43, 91, 92, 67, 266, 265].

In this section, we first give approximate boundaries between "feasible" and "not feasible" attacks. We then describe generic preimage and collision attacks, attacks on iterated hash functions, and differential cryptanalysis. Finally, we argue on the utility of "random-looking" colliding messages and discuss some attacks targeting properties beyond the preimage and collision resistance.

2.5.1 Complexity limits for the feasibility of attacks

The boundary between a "feasible" attack and a "non-feasible attack is not precise as it strongly depends on the resources the attacker is up to invest in order to find collisions. The feasibility is usually estimated in terms of time and memory complexities, to which we add the lengths of hashed messages.

Today in 2009, attacks running in time 2^{60} are feasible by large computers clusters and attacks in time 2^{80} are believed to be infeasible. According to Lenstra and Verheul's recommendations for the key lengths of symmetric cryptographic [164, 163], the current limit is 74 bits if we believe that DES was secure until 1982, which it was with respect to most adversaries but probably not for large companies and governmental agencies. In the table below, we show their recommendations for the years 2009-2016, assuming DES was secure until 1982 or 1980 [112].

Year	2009	2010	2011	2012	2013	2014	2015	2016
Hyp. 1982	74	75	76	77	77	78	78	79
Hyp. 1980	76	76	77	78	78	79	80	80

Setting apart all the engineering problems of managing a huge cluster of memory disks, the practical limit for memory can be first evaluated by its cost. As a memory hard disk of 1TB (2^{40}) costs today about 100 €, we may estimate that attacks with memory requirements above 2^{65} or 2^{70} are not feasible today even for big governmental agencies (as an example, in 2008 the CERN had a distributed computing and data storage infrastructure to store annually 15PB (2^{57}) of data from the Large Hadron Collider [6]).

In practice, memory access is often more expensive than memory itself. Processing data on an external disk is considerably slower than processing data on RAM. Therefore, an attack requiring to read and write 2^{60} bits of data may already be considered infeasible even for big governmental agencies, simply because the memory accesses will increase a lot the execution time.

Besides these two standard criteria, we will consider a third feasibility criterium of attacks, namely the size of the messages produced. For the Cayley hash functions discussed in this thesis, there exist trivial collision attacks that are efficient both in time and memory but still not practical as the colliding messages they produce have size far too large for any reasonable application. In most applications, hash functions process messages of a few kb or Mb; when large disks are hashed it may make sense to consider messages of size 2^{40} to 2^{50}. On the other hand, messages of size 2^{60} will never be hashed.

2.5.2 Generic attacks

Exhaustive search and random trials. Given the value h of a randomly chosen message, an adversary can find a preimage m such that $H(s,m)=h$ (where the key is known by the adversary) by trying successive values $m = 1, 2, ...$ until he finds a preimage. If the output is of λ size, the attack requires expected time 2^λ. Finding preimages of various hash values h_i requires essentially the same time if the key is not changed for each computation. Alternatively, the adversary can test random messages among some finite set larger than 2^λ and also succeed in expected time 2^λ.

Birthday attack. For a group of 23 people, the probability that at least two persons were born on the same day is larger than 1/2. As typically 365 persons are supposed to be needed, this phenomenon is called *the birthday paradox* even if it is not strictly paradoxical. Similarly, if the codomain of a practical hash function is of 2^λ size, an adversary can find collisions after $2^{\lambda/2}$ hash computations on random messages.

Indeed, let $N = 2^\lambda$ the number of output values. The probability to find collisions after $N' = 2^{\lambda/2}$ random trials is

$$\begin{aligned}\Pr[\texttt{col}] &= 1 - \frac{N}{N}\frac{N-1}{N}...\frac{N-N'+1}{N}\\ &= 1-\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)...\left(1-\frac{N'-1}{N}\right).\end{aligned}$$

Using Taylor's first order approximation $e^x \approx 1 + x$ we get

$$\Pr[\texttt{col}] \approx 1 - e^{-\frac{1}{N}-\frac{2}{N}-...-\frac{-N'+1}{N}} \approx 1 - e^{-\frac{N'^2}{2N}} \approx 0.4.$$

The birthday attack was first pointed out by Yuval [273]; it requires time and memory $2^{\lambda/2}$. The memory requirements were dropped to negligible by Quisquater and Delescaille [219] by translating the collision problem to the problem of detecting cycles in an iterative mapping. In the modified birthday attack, instead of choosing the messages randomly, the adversary chooses them deterministically according to the previous hash value. This induces a deterministic mapping on a finite set that will eventually repeat, therefore producing cycles. The advantage of this approach is that there is no need to store all the hash values; the adversary may store only *distinguished points*, for example those values beginning with a large number of zero bits. The modified birthday attack requires essentially the same $2^{\lambda/2}$ time and a negligible memory. Moreover, a parallel version has been given by van Oorschot and Wiener [261]. Other techniques for removing requirements in cycle-detection methods are reviewed by Shamir in [242].

2.5.3 Attacks on iterated hash functions

Hash functions that iterate a compression function, for example with the Merkle-Damgård transform, are sensitive to more efficient attacks than the generic attacks.

Meet-in-the-middle attack. If the compression function is invertible, preimages can be computed in a time roughly $2^{\lambda/2}$ by extending the birthday attack as follows: apply the compression function to $2^{\lambda/2}$ random messages and apply it backward to $2^{\lambda/2}$ other random messages. By the birthday paradox, there is a large probability that the adversary finds a common value "in the middle". The attack also has a memory-free version [220]; it is not feasible if the compression function is preimage resistant.

Fixed point attack. The idea of this attack is to look for an intermediate value h_{i-1} and a message block m_i such that $f(s, h_{i-1}||m_i) = h_{i-1}$. The attack allows inserting any number of blocks m_i without changing the hash value. In a Merkle-Damgård construction, it becomes very practical if the IV can be selected by the adversary.

Multicollision attack. By the birthday paradox, a collision can be found to a compression function in time $2^{\lambda/2}$. Repeating the collision search $\lambda/2$ times, an adversary can find message blocks $m_1, m_1', m_2, m_2', ..., m_{\frac{\lambda}{2}}, m'_{\frac{\lambda}{2}}$ such that

$$\begin{array}{rcl} h_1 & := & f(s, h_0||m_1) = f(s, h_0||m_1') \\ h_2 & := & f(s, h_1||m_2) = f(s, h_1||m_2') \\ & \dots & \\ h_{\frac{\lambda}{2}} & := & f(s, h_{\frac{\lambda}{2}-1}||m_{\frac{\lambda}{2}}) = f(s, h_{\frac{\lambda}{2}-1}||m'_{\frac{\lambda}{2}}). \end{array}$$

These message blocks can be combined into $2^{\lambda/2}$ messages of $\lambda/2$ blocks that hash to the same value. Finding these "multicollisions" hence requires time only $\frac{\lambda}{2}2^{\lambda/2}$ while on an ideal function it would require a time $2^{\lambda(2^{\lambda/2}-1)/2^{\lambda/2}} \approx 2^{\lambda}$.

This observation has been used by Joux [145] to improve the birthday attack on a class of hash functions. Suppose $G, H : \{0,1\}^{\kappa} \times \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ are two hash functions with ideal collision-resistance (meaning that the best attack has expected time $2^{\lambda/2}$). If G and H were ideal, the function $F(s,m) := G(s,m)||H(s,m)$ would have ideal collision-resistance in 2^{λ}. However, if G is an iterated hash function, an adversary can construct $2^{\lambda/2}$ collisions for G in time $\frac{\lambda}{2}2^{\lambda/2}$. By the birthday paradox, these $2^{\lambda/2}$ messages are likely to give one collision for H, hence for F.

2.5.4 Differential cryptanalysis

After Biham and Shamir introduced it for key recovery on block ciphers [43], differential cryptanalysis has also been applied to stream ciphers and to most dedicated hash functions. As stated by Dobbertin [91], *the basic idea is that a (small) difference between only one of the input variables can be controlled in such a way that the differences occurring in the computations of the two associated hash values are compensated for at the end.* The art of differential cryptanalysts is to find good *differential paths* through the whole algorithm computation.

Most attacks on dedicated hash functions use extensions of this idea [91, 92, 67, 266, 265]. Differential cryptanalysis has been applied mainly on unkeyed, Merkle-Damgård-based hash functions to find collisions of the form $f(s,h||m) = f(s,h||m')$ on the compression function. As unkeyed hash functions are targeted, the key s can be considered as fixed. The value h is considered as random, which is realistic because in a Merkle-Damgård construction it is the output of the compression function from the previous round.

The compression functions of recent hash algorithms have a lot of rounds that improve the bit interdependencies through non-linear functions; it is therefore no longer possible to find a full differential path resulting in a collision for the compression function with a probability 1. However, the cryptanalyst can search in all stages of the algorithm for particular differences in the input bits that with a large probability (on the value h) will result in small differences a few stages later. Combining these *differentials* appropriately to cancel differences may lead to a collision at the end of the hash computation with some probability: when the attack is repeated, it is likely to succeed after a time inversely proportional to this probability.

Near-collisions (messages whose hash values differ by only a few bits) are also targeted by differential attacks, in which case a full collision is obtained by iterating the attack, to produce a pair of colliding messages whose lengths are a few block long [42].

2.5.5 On "meaningful" collisions

When a new collision attack is found, the messages it produces most often have the form of apparently random bits. These collisions have no chance to be meaningful for any high-level application of hash functions so at first sight they should not threaten the security of these applications.

This opinion is erroneous in two ways. First, attacks get improved over the years; they become faster and allow for more structure on the colliding messages. Second, collisions *are* meaningful in some applications, and meaningless collisions can often be turned into useful ones: Lucks and Daum have shown how to build two PS files with the same signature using MD5 collisions [169], Gebhardt et al. have extended their ideas to PDF, TIFF and Word97 formats [110] and Stevens et al. have produced colliding MD5 based X.509 certificates [252].

2.5.6 Beyond collision resistance

Attacks on hash functions have been mainly targeting their preimage, second preimage and collision resistance because these are the most popular security requirements for hash functions. For applications requiring PRFs or random oracles there exist many more kinds of attacks, sufficient to break some cryptographic schemes.

The most popular one targets near-collision resistance: a near-collision is made of two messages that produce the same hash value up to a few bits. Clearly, near-collisions may break the security of schemes that truncate the output of the hash function before usage. Near-collisions on a compression function can also sometimes be turned into a full collision for the hash function [42].

Figure 2.5: Hash functions are an invaluable tool for such a wide range of cryptographic applications that they have been compared to Swiss army knives

Hash functions are often used to destroy the algebraic structure of other schemes. To this end, they should of course not present the same structure. A hash function with a complementarity property used with DES, or a function with multiplicative properties used in the RSA signature protocol, would introduce significant security threats [27]. On the other hand, many hash functions that are provably collision resistant have this kind of weaknesses because of their mathematical structure. We will further discuss these *malleability* properties in Chapter 8.

2.6 Applications

Hash functions are an invaluable tool for such a wide range of cryptographic applications that they have been compared to Swiss army knives [101].

The main usages of hash functions are message authentication codes (MACs) and digital signature algorithms but they have also been used (among other applications) to build commitment schemes and pseudorandom number generators, for entropy extraction, and to protect password databases.

2.6.1 Message authentication codes

Message authentication codes (MAC) are a fundamental tool to guaranty the *integrity* of documents. Suppose a bank receives the following order from Alice "*Send* €1000 *to Bob*". The bank needs to be sure that the message has not been modified: for example, that Bob did not intercept a message saying "*Send* €10 *to Bob*" or "*Send* €1000 *to Charly*" and modify it in its interest before forwarding it to the bank. Message authentication codes can be used to prevent such attacks.

A message authentication code is made of three algorithms. The *key-generation algorithm* produces a secret key s to be shared between Alice and the bank; the *tag-generation algorithm* produces a tag t from a message m and a key s; finally, the *verification algorithm* receives a key s, a message m and a tag t, and decides whether t is a valid tag or not.

A MAC is secure if it is *existentially unforgeable under an adaptive chosen-message attack*: no adversary, after receiving valid tags on any message of his choice, should be able to produce a valid tag on a new message of his choice [37].

Definition 2.11 *A* message authentication code *(or MAC) is a triple of* PPT *algorithms $\mathcal{M} = (Gen, Mac, Ver)$ such that*

- *The* key-generation algorithm *Gen takes as input 1^n and outputs a key s with $|s| \geq n$;*
- *The* tag-generation algorithm *Mac takes as input a key s and a message $m \in \{0,1\}^*$, and outputs a tag t;*
- *The deterministic* verification algorithm *Ver takes as input a key s, a message m and a tag t. It outputs a bit b, with $b = 1$ meaning* valid *and $b = 0$ meaning* invalid.

Moreover, it is required that for every n, every key s that has been output by $Gen(1^n)$, every $m \in \{0,1\}^*$, and every t that has been output by $Mac(s, m)$, we have $Ver(s, m, t) = 1$.

The triple of PPT algorithms (Gen, Mac, Ver) is called a *fixed-length MAC for messages of length μ* if the algorithm Mac is only defined for messages of length μ, and the algorithm Ver outputs 0 on messages of a different size.

Definition 2.12 *A MAC $\mathcal{M} = (Gen, Mac, Ver)$ is* secure *or* existentially unforgeable under adaptive chosen-message attacks *if for all* PPT *algorithms A, the probability*

$$Adv_{\mathcal{M}}^{\mathtt{Forge},A}(n) := \Pr\left[Exp_{\mathcal{M}}^{\mathtt{Forge},A}(n) = 1\right]$$

is negligible, where $Exp_{\mathcal{M}}^{\mathtt{Forge},A}$ is defined below.

Experiment $Exp_{\mathcal{M}}^{\texttt{Forge},A}(n)$:

- a key s is generated by running Gen on input 1^n;
- the adversary A has an oracle access to $Mac(s, .)$: he sends queries m_i of his choice and receives the corresponding $Mac(s, m_i)$;
- the adversary A outputs a pair (m, t);
- $Exp_{\mathcal{M}}^{\texttt{Forge},A} = 1$ if $Ver(s, m, t) = 1$ and the adversary had not queried his oracle on m.

Unlike hash function keys, MAC keys are secret values. Secure fixed-length MACs can be constructed from PRFs, and used in turn to construct secure MACs; we refer to [150] for more details. However, there exists no construction of a secure MAC based on CRHF.

The most common constructions of message authentication code are the HMAC and NMAC of Bellare, Canetti and Krawczyk [33, 34] that have been incorporated into an American federal standard [15].

The construction NMAC uses two random keys k_1, k_2, a compression function f and the Merkle-Damgård transform as follows. First, the Merkle-Damgård transform of the message is computed with k_2 as initial value; then, the result is passed again to the compression function together with the other secret key k_1 (see Figure 2.6.1).

A fixed-length variant of this construction is obtained by replacing the Merkle-Damgård transform by a single compression function evaluation. The security of NMAC is *not* implied by the collision-resistance of f; some additional pseudo-randomness properties are at least necessary. However, NMAC is a secure MAC if its fixed-length variant is a secure MAC and the compression function is collision-resistant [150].

The HMAC construction can be seen as a particular instance of NMAC, where the keys k_1 and k_2 are generated from a single key k as follows

$$k_1||k_2 := G(k) = f(s, IV||(k \oplus \texttt{opad}))||f(s, IV||(k \oplus \texttt{ipad})).$$

The value `opad` is defined as the byte “0x36” repeated as many times as needed; the value `ipad` is defined similarly with the byte “0x5C”. The advantage of HMAC over NMAC is that it can be defined from any iterated hash function without changing its initial value. It is secure if the corresponding NMAC construction is secure and the output of G is “indistinguishable from random bits” (that is, if G is a pseudorandom number generator) [150].

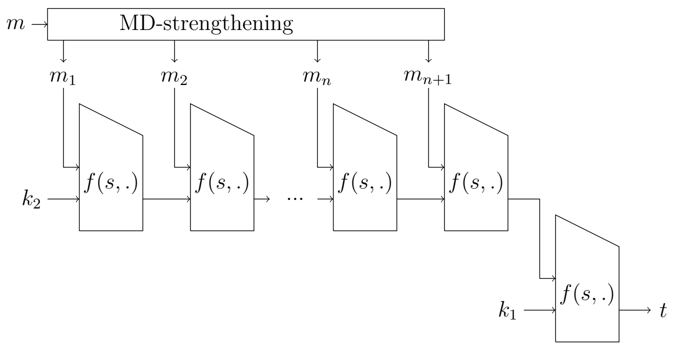

Figure 2.6: NMAC construction from a compression function f

Alternatively to NMAC and HMAC, Black et al. have proposed UMAC, a construction based on a pseudorandom function (in practice, a block cipher or a hash function) and a universal hash function. The construction is much faster than NMAC and HMAC and is secure if the block cipher or the hash function used is a pseudorandom function [45]. UMAC has also been standardized but is much less popular than HMAC today. Constructions based on block ciphers are discussed in Section 3.4 and previous constructions are reviewed in [217, 218].

2.6.2 Digital signatures

Like message authentication codes, digital signatures can be used to ensure integrity of documents. Digital signature schemes can be constructed from CRHF or UOWHF; they are typically much slower to compute than MACs but present a few practical advantages.

Digital signatures are public key primitives: each signer has a pair of keys (pk, sk). The *secret key sk* is used by its owner to sign messages, and the *public key pk* is used to verify a signature. While the verification of a MAC requires the knowledge of a secret key, a signature is publicly verifiable hence no preliminary key agreement is necessary, the signature may be transferred and it cannot be repudiated [150].

The definition and the security model of a digital signature scheme are very close to those of a MAC.

Definition 2.13 *A* signature scheme *is a triple of* `PPT` *algorithms* $\mathcal{S} = (Gen, Sig, Ver)$ *such that*

- *The* key-generation algorithm Gen *takes as input* 1^n *and outputs a pair of (public, private) keys* (pk, sk)*;*
- *The* signing algorithm Sig *takes as input a private key* sk *and a message* $m \in \{0,1\}^*$*, and outputs a signature* σ*;*
- *The deterministic* verification algorithm Ver *takes as input a public key* pk*, a message* m *and a signature* σ*. It outputs a bit* b*, with* $b = 0$ *meaning* valid *and* $b = 1$ *meaning* invalid.

Moreover, it is required that for every n*, every* (pk, sk) *output by* $Gen(1^n)$*, every* $m \in \{0,1\}^*$*, and every* σ *that has been output by* $Sig(sk, m)$*, we have* $Ver(pk, m, \sigma) = 1$*.*

The triple of `PPT` algorithms (Gen, Sig, Ver) is called a *fixed-length signature scheme for messages of length* μ if the algorithm Sig is only defined for messages of length μ, and the algorithm Ver outputs 0 on messages of a different size.

Definition 2.14 *A signature scheme* $\mathcal{S} = (Gen, Mac, Ver)$ *is* secure *or* existentially unforgeable under adaptive chosen-message attacks *if for all* `PPT` *algorithms* A*, the probability*

$$Adv_{\mathcal{S}}^{\texttt{Forge},A} := \Pr\left[Exp_{\mathcal{S}}^{\texttt{Forge},A} = 1\right]$$

is negligible, where $Exp_{\mathcal{S}}^{\texttt{Forge},A}$ *is defined below.*

Experiment $Exp_{\mathcal{S}}^{\texttt{Forge},A}$:

- $Gen(1^n)$ is run to obtain keys (pk, sk);
- the adversary A has oracle access to $Sig(sk, .)$: he sends queries m_i of his choice and receives the corresponding $Sig(sk, m_i)$;
- the adversary A outputs a pair (m, σ);
- $Exp_{\mathcal{S}}^{\texttt{Forge},A} = 1$ if $Ver(pk, m, \sigma) = 1$ and the adversary had not queried his oracle on m.

In theory, the usual way to build a secure signature scheme is to first build a secure *one-time signature scheme*, which is a signature scheme satisfying the weaker security property that the adversary is not able to forge a signature if he queries his oracle only once [159]. The construction leads to secure schemes from collision-resistant hash functions, but the schemes are not efficient at all. In practice, many signature schemes are constructed using either the *hash-and-sign paradigm* or the *Fiat-Shamir transform*, resulting in efficient schemes but only secure in the random oracle model.

A secure signature scheme can be built from a secure one-time signature scheme by refreshing the keys: at each signature, a new public key is issued and signed together with the message. One-time signature schemes can be constructed with one-way functions [159] hence from CRHF (because collision resistant hash functions are one-way functions). Naor and Yung have shown that Universal One-Way Hash functions are actually sufficient [192].

The hash-and-sign paradigm is to construct signature schemes by first hashing the messages then signing them with a fixed-length signature scheme.

Let $\mathcal{S} = (Gen_{\mathcal{S}}, Sign, Ver)$ be a fixed-length signature scheme and $\mathcal{H} = (Gen_{\mathcal{H}}, H)$ be a hash function. The *hash-and-sign paradigm* is to construct a signature scheme $\mathcal{S}' = (Gen', Sign', Ver')$ as follows

- Gen': on input 1^n, run $Gen_{\mathcal{S}}$ to obtain (pk, sk) and run $Gen_{\mathcal{S}}$ to obtain s, then return $(pk||s, sk||s)$;
- Sig': on input a private key $pk||s$ and a message $m \in \{0,1\}^*$, return $\sigma = Sig(sk, H(s,m))$;
- Ver': on input a public key $pk||s$, a message $m \in \{0,1\}^*$ and a signature σ, output 1 if and only if $Ver(pk, H(s,m), \sigma) = 1$.

The hash-and-sign paradigm extends the domain of $\mathcal{S}$ and increases its speed without weakening its security. Signing very long messages would require very large parameters if they were not hashed first, with a catastrophic loss in efficiency. The signature scheme $\mathcal{S}'$ is secure if $\mathcal{S}$ is secure and $\mathcal{H}$ is collision resistant [150]. Actually, it can also be secure when $\mathcal{S}$ is not secure in the sense of Definition 2.14 but $\mathcal{H}$ is modeled as a random oracle, an argument that justifies constructions like RSA, Paillier or Rabin full-domain signatures [228, 200, 222, 33] and DSS [12].

Fiat and Shamir have shown how to transform any three-round identification scheme into a signature scheme secure in the random oracle model

[100]. Roughly, in a three-round identification scheme a *prover* uses his secret key to prove his identity to a *verifier* as follows: the prover sends a message CMT called *commitment* to the verifier, the verifier returns a random string CH called *challenge*, and the prover provides a *response* RSP. The verifier applies a verification algorithm to the prover's public key and the conversation $CMT||CH||RSP$ to decide whether or not he believes his interlocutor is the person he claims to be. The security of an identification scheme requires that the verifier always believes an honest prover and would believe a cheater only with a negligible probability.

The Fiat-Shamir transform produces a signature scheme in which the random challenge is replaced by a hash computation of the commitment. To sign a message m using a public hash function $\mathcal{H} = (Gen, H)$ (with fixed key s), the signer generates a commitment CMT as the prover would, he computes $CH = H(s, CMT||m)$, he computes a response RSP to CH as the prover would and returns the signature $CMT||RSP$. The verifier computes $CH = H(s, CMT||m)$ and would accept the signature if the verifier of the identification had accepted the conversation $CMT||CH||RSP$. For further details and formalism, we refer to [100, 198].

The Fiat-Shamir transform produces secure signature schemes in the random oracle model if the identification scheme is secure against *passive* attacks (that are attacks in which the adversary may read all the messages but not modify any of them) and the commitments are drawn at random from a large space [17]. Most notably, Schnorr [237] and GQ [124] signatures are constructed that way. The Fiat-Shamir transform also produces *forward-secure* signature schemes (schemes for which the security of previous signatures remains intact if the secret key leaks today) from forward-secure identification schemes [17].

2.6.3 Other applications

Hash functions are used to guaranty the integrity of documents via message authentication codes and digital signatures, but they have many other applications; we present some of them in this section.

Commitment schemes. Commitment schemes allow a user to commit on a bit value b (the *commitment*) that is hidden, and to reveal this bit later. It is the equivalent of a sealed box sent without key: its content is hidden until the sender send the key to the receiver. Commitment schemes are required to be both hiding and bidding: no receiver can guess the commitment but with a probability negligibly larger than 1/2, and no sender can produce a commitment valid for both values of the bit.

Commitment schemes have been first introduced by Brassard et al. [56]; they have applications for electronic coin flipping [48], zero-knowledge proofs of knowledge [118] and in verifiable secret sharing. Commitment schemes can be constructed from pseudorandom number generators [191] and from one-way functions [126], hence from CRHF.

In the random oracle model, a trivial construction of a commitment scheme would be to send as commitment the hash of a random even value to commit to 0 and the hash of a random odd value to commit to 1. This solution is not secure for a CRHF because nothing in the definition of a CRHF prevents the last bit of the message from leaking. The scheme would however be secure with a perfectly one-way probabilistic hash function [61]. In [127], a practical solution is proposed that uses CRHF and a particular instance of universal hash functions.

Pseudo-random number generation. Pseudorandom number generators (PRNG) receive as input a *seed* or small truly random bit sequence r, and output a larger bit sequence that can be distinguished from a truly random bit sequence by no `PPT` algorithm but with negligible probability. Assuming the hash function $\mathcal{H} = (Gen, H)$ behaves "sufficiently randomly" (in particular, if it behaves like a random oracle), the following constructions are good PRNGs

$$\begin{aligned} G_{\text{counter}}(r) &:= H(s,r)||H(s,r+1)||H(s,r+2)||... \\ G_{\text{serie}}(r) &:= H(s,r)||H(s,H(s,r))||H(s,H(s,H(s,r)))||... \end{aligned}$$

where for seeds of size μ, the symbol $+$ represents addition modulo 2^{μ}. Other constructions are also possible [8].

One important application of these generators are *key derivation techniques*. Secret keys are very important to guarantee the security of cryptographic protocols. Using one single secret key in many protocols would be a dangerous practice: using this key in unappropriate ways or in an insecure protocol only once would be enough to reveal it, therefore threatening the security of all the other protocols that use the same key. For this reason, the private keys used in protocols are usually *derived* from a "master" private key, using for example a hash function in counter mode as above [8]. Because the hash computation cannot be reversed and its output is "perfectly random", the leakage of one particular secret does not disclose any partial information on the master key.

Entropy extraction. Many applications in cryptography require good long secret pseudorandom bit sequences of one hundred or one thousand bits but no human being can remember such a long sequence of bits. On the other hand, a human can remember one thousand bits of information if this information is structured like a passphrase but passphrases have a low entropy that makes them unsuitable for many applications. In this context, universal hash functions can be used to transform long passphrases into short bitstrings with maximal entropy. Similarly and even more importantly, at the end of a key agreement protocol like Diffie-Hellman [90], a hash function is used to transform a mathematical object with a good entropy (typically an element of a large group) into a bitstring that will serve as an AES key.

Formally, let $\mathcal{D}$ be a probability distribution on a set S. The *Renyi entropy* of $\mathcal{D}$ is the probability that $x = y$ when x and y are drawn independently and according to distribution $\mathcal{D}$:

$$\mathbf{H_{ren}}(\mathcal{D}) = -\log \Pr_{x,y \overset{\mathcal{D}}{\leftarrow} S}[x = y].$$

If $\mathcal{D}_1$ and $\mathcal{D}_2$ are probability distributions on a same set S, the statistical distance between them is

$$\mathbf{L_1}(\mathcal{D}_1, \mathcal{D}_2) = \sum_{x \in S} \frac{|\Pr_{\mathcal{D}_1}[x] - \Pr_{\mathcal{D}_2}[x]|}{2}.$$

The so-called “left-over hash lemma” shows that universal hash functions can be used to “smooth” probability distributions to the uniform distribution. This lemma has various versions with the same intuitive meaning; we adapt the version of [128] to our notations.

Lemma 2.1 (Left-over Hash lemma) *Let $\mathcal{D}_n$ be a probability distribution on $\{0,1\}^n$ that has a Renyi entropy at least $r(n)$. Let $H : \{0,1\}^{\kappa(n)} \times \{0,1\}^{\mu(n)} \to \{0,1\}^{\lambda(n)}$ be a strongly universal hash function. Let $e(n) := (r(n) - \lambda(n))/2$, let $\mathcal{U}_{\lambda(n)}$ be the uniform distribution on the set $\{0,1\}^{\lambda(n)}$ and let $\mathcal{D}_H^n$ be the distribution of the output of H when its key is chosen uniformly in $\{0,1\}^{\kappa(n)}$ and its message is chosen according to distribution $\mathcal{D}_n$. Then $\mathbf{L_1}(\mathcal{U}_{\lambda(n)}, \mathcal{D}_H^n) \leq 2^{-(e(n)+1)}$.*

In practice, a standardized hash functions like SHA will be used in this application.

Password checking. Hash functions are used in server-client applications where the clients identify to their server with a password. To be able to verify the passwords, the server must have somehow stored them in a database but this introduces serious security threats: a legitimate employee or a hacker who would enter this database could read all the passwords and later use them. A classical solution to this problem is to store a hash value of each password instead of the passwords themselves: in the authentication phase, the server re-computes this hash value from the password introduced, and compares it to the value in its database. Intuitively, the previous attack is avoided if the hash function is preimage resistant.

2.7 Further readings

Hash functions are a fundamental tool in cryptography, discussed in all cryptography textbooks (among which [114, 150, 176, 235]) and in three surveys [31, 214, 187]. No survey has appeared recently, but Bart Preneel is writing a book entirely dedicated to the topic that should appear in 2009. The most important publications on hash functions are [83, 65, 212, 61, 63, 232].

The security definitions of hash functions differ from one application to the other. We have given in Section 2.2 the main notions of preimage, second-preimage and collision resistance, and in Section 2.3 the notions of universal hash function, of pseudorandom function and of random oracles. The notion of non-malleability will be further discussed in Chapter 8 of this thesis; references for other security notions are given in the appendix A of [232]. Trapdoor hash functions [156, 244, 211] will be introduced in Section 3.1.1.

The Merkle-Damgård transform presented in Section 2.4 is used in most "custom designed" hash functions like the MD_+ and SHA families. It is beyond the scope of this thesis to discuss these functions and the large number of new algorithms that have recently been proposed for the NIST competition [1]. We refer to the ECRYPT "SHA-3 Zoo page", the "Hash Function Lounge" of Paulo Barreto and the bibliography maintained by Søren Steffen Thomsen [16, 2, 3] for pointers to the specifications of these algorithms and up-to-date information on their security.

Chapter 3

The reductionist approach to cryptographic hash functions

Collision resistance is the most important security property for hash functions. The notion is intuitive, it is a necessary requirement for many applications and it also implies preimage resistance and second preimage resistance.

Collision resistant hash functions can be constructed under widely believed number theoretic assumptions like the discrete logarithm and the factorization assumptions, but these constructions only give fixed-length hash functions and the corresponding hash algorithms have traditionally been very slow. Indeed, until recently cryptographic hash functions could be divided into number theory-based inefficient hash functions and heuristic, specially designed, efficient hash functions.

Throughout this thesis, a *provable hash function* is a hash function whose collision resistance is implied by a hardness assumption on a (known) mathematical problem. In particular, the word "provable" does not refer to any additional property of hash functions like universality or pseudorandomness. The theory of cryptography has provided provably secure constructions for pseudorandom functions but these constructions are still far less efficient than provably collision-resistant hash functions. The functions described in this chapter have only been designed to be provably collision resistant; they often satisfy extra properties of universality but most of them are not good candidates of pseudorandom functions.

Provable hash functions usually have a rich mathematical structure that implies homomorphic properties and weak behaviors on particular inputs. These properties do not contradict collision resistance but prevent the hash functions from being used in applications requiring more properties. In Chapter 9, we will show how to modify the Zémor-Tillich hash function to *heuris-*

tically satisfy good pseudorandom properties while preserving its *provable* collision resistance.

Collision resistance is of course only guaranteed if the underlying mathematical problem is really difficult. As discussed in Chapter 2, a problem is hard if no probabilistic polynomial-time algorithm can solve it. For instance, it is widely believed that the factorization problem, the discrete logarithm problem and the elliptic curve discrete logarithm problem are hard. The first provable cryptographic hash functions were collision resistant under a hardness assumption on one of these problems but they were very inefficient compared to specially designed hash functions.

The new generation of provable hash functions are much faster but their collision resistance relies on less standard assumptions. To be meaningful, a hardness assumption must have been carefully examined by the designers of the hash function who must provide some “evidence” of its validity. Ideally, the assumption should have been studied before in other contexts which can be referred to as “evidence” of hardness; the underlying mathematical problem should have a clear and concise form that facilitates its examination by mathematicians even outside the cryptographic community.

This chapter reviews some provable hash functions. Section 3.1 gives some early proposals; Sections 3.2 and 3.3 detail VSH and SWIFFT; Section 3.4 and 3.5 give the main results on block cipher-based hash functions and expander hash functions.

Among the “provable” hash functions that are reasonably efficient, VSH is certainly the function whose security is the most convincing: a collision algorithm on VSH would probably imply significant improvements on current factorization algorithms. The collision resistance of SWIFFT reduces to a particular class of a lattice problem whose general formulation is NP-hard. The security of block cipher-based hash functions is proved under some idealization of the underlying block cipher. Finally, the security of expander hashes depends on each particular instance: while LPS and Morgenstern hashes are now completely broken, the Zémor-Tillich hash function has not been seriously damaged since 1994.

“Provable security” with respect to non-standard assumptions may not give *today* the same confidence as a security reduction to the factorization problem, but the confidence grows over the years if the underlying problems are studied and the function does not get broken. In custom designed hash functions, security flaws may be well-hidden and missed by the first design evaluations and only appear after validation. The security reduction, however, provides a concise well-defined mathematical problem to solve in order to break the function; the failures of cryptanalysts over the years bring significant evidence for the hardness of the problem.

3.1 Early instances of CRHF

The first constructions of collision resistant hash functions rely on the factorization problem, the discrete logarithm problem or on certain classes of knapsack problems. Many constructions use the Merkle-Damgård transform (see Section 2.4 of Chapter 2) or any other transform to extend a fixed-length hash function into an arbitrary length hash function. Alternatively, some factorization and discrete logarithm-based hash functions use *claw-free functions* recursively.

3.1.1 Factorization-based hash functions

The integer factorization problem is the following:

Problem 3.1 (Integer Factorization Problem or IFP) *Given $n = pq$ for two randomly chosen primes p and q of the same size, find the prime factors p and q.*

The factorization problem has been studied for ages, probably more than 4000 years. The best algorithms today are index-calculus and sieving algorithms; they both run in subexponential time, which is better than trivial exhaustive searches but still far from PPT . The factorization problem is believed to be infeasible today for composite n of at least 1350 bits [112].

Collision resistant hash functions exist if the factorization problem is hard. The construction of Goldwasser et al. [120, 121] uses a *claw-free function* in a recursive way.

Definition 3.1 *A k-*claw-free function *is a pair of* PPT *algorithms (Gen, H) such that*

- ***Polynomial-time indexing:*** *$Gen(1^n)$ returns $(f_0, f_1, ..., f_k)$ where the f_i are the descriptions of permutations on a same domain D that are chosen uniformly randomly in the set of all permutations of domain D (we suppose that the description of D is implicitly contained in the description of the permutations) ;*

- ***Polynomial-time evaluation:*** *H receives the description of a permutation f_i and an element m of its domain, and returns $f_i(m)$;*

- ***Claw-freeness:*** *for all $(f_0, f_1, ..., f_k)$ generated by Gen, no* PPT *algorithm can create a* claw *that are two values s, s' such that there exist $i \neq i'$ with $f_i(s) = f_{i'}(s')$.*

If factoring is hard for integers $n = pq$ with primes $p \equiv 3 \bmod 8$ and $q \equiv 7 \bmod 8$, then the choice $f_0(s) = s^2 \mod n$, $f_1(s) = 4s^2 \bmod n$ and D the set of quadratic residues modulo n, leads to a 2-claw-free function [120]. Indeed, if $f_0(s) = f_1(s')$ then with a probability 1/2 either $\gcd(s + 2s', n)$ or $\gcd(s - 2s', n)$ is p or q. The computation of a hash value requires roughly one modular squaring per bit of message.

The construction of Goldwasser et al. uses a 2-claw-free function as described in the following theorem and represented in Figure 3.1.

Theorem 3.1 *Let $\mathcal{P} = (Gen_{\mathcal{P}}, H_{\mathcal{P}})$ be a 2-claw-free function. Then the hash function $\mathcal{H} = (Gen_{\mathcal{H}}, H_{\mathcal{H}})$ is a collision resistant hash function, where $Gen_{\mathcal{H}}$ and $H_{\mathcal{H}}$ are defined as follows:*

- *$Gen_{\mathcal{H}}$: on input 1^n, runs $Gen_{\mathcal{P}}(1^n)$ to get (f_0, f_1); then selects uniformly randomly an element s in the domain and returns (s, f_0, f_1).*

- *$H_{\mathcal{H}}$: on input (s, f_0, f_1) and m; uses a* prefix-free encoding[1] *to encode m in binary and gets a bit sequence $m_0m_1...m_N$; then returns*

$$f_{m_N}(f_{m_{N-1}}(...f_{m_0}(s)...)).$$

Damgård has given two variants of Goldwasser et al.'s scheme that are also based on factorization assumptions [82]. The first variant uses moduli $n = p_1...p_k$ with more prime factors and the permutations $f_i(s) = (a_i s)^2 \bmod n$ for arbitrary $a_i \in \mathbb{Z}_n^*$ and quadratic residues s. The computation of a hash value requires only one modular squaring for $\log_2 k$ bits of message but the modulus n must be larger to guarantee the same security level.

The second variant defines $f_i(s) = a_i s^2 \bmod n$ where the a_i are randomly chosen quadratic residues modulo n. The modulus n may be kept as before and $\log_2 k$ bits are processed at once but the security of this scheme is slightly different from the previous ones: the resulting hash function is collision resistant if no algorithm can compute a square root modulo n of one of the values $a_i a_{i'}^{-1}$ for $i \neq i'$. In this scheme unlike in the previous ones, it is crucial

[1] A prefix-free encoding is an encoding such that for all valid encoding $m = m_1m_2...m_N$, no valid encoding can be formed by appending symbols at the end of m. One prefix-free binary encoding is the following: code 1 by 11, 0 by 00, and terminates all other encodings with 01.

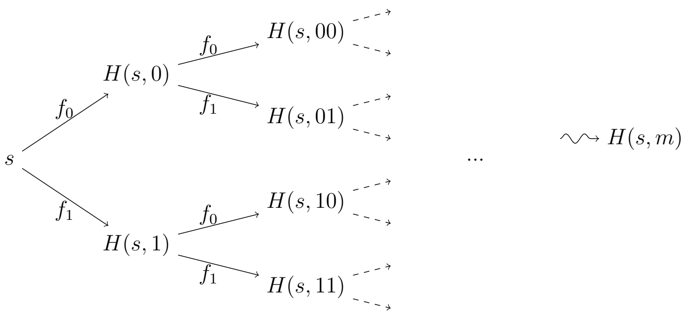

Figure 3.1: The Goldwasser et al.'s construction on pre-encoded messages

to randomly choose the a_i as some choices like $a_2 = 4a_1$ lead to an insecure function.

Another scheme secure if factorizing is hard has been given by Shamir and Tauman [244]. In this scheme, the message is put on the exponent rather than on the basis: for a large composite number $n = pq$ and a generator g of $\mathbb{Z}_n^*$, the hash value of a message m is defined as $g^m \bmod n$. The computation of a hash value requires one and a half modular multiplication per bit of message. The scheme is collision resistant as for any collision (m, m'), the value $m - m'$ must be a multiple of $\varphi(n) = (p-1)(q-1) = n-p-q+1$ hence reveal the factorization because p and q are roots of $X^2-(n+1-\varphi(n))X+n$.

All these hash functions based on factorization are *trapdoor* hash functions [156], in the sense that collisions can actually be computed efficiently but only with the help of some information called a *trapdoor* (here the trapdoor is the factorization).

At first sight, the existence of a trapdoor is a weakness, but its security damage is limited and can be easily removed. If a factorization-based trapdoor hash function is used in a signature algorithm, the person who chooses the modulus n should not be allowed to use it because he is actually able to produce two colliding messages m, m', to sign m and then pretend he signed m'. Although such a collision would immediately cast suspicion on the signer because it would almost surely reveal the factorization of n, it is best preventing the attack by letting the modulus be generated by a trusted party or by a group of users using some secure multiparty computation [52].

To conclude this discussion on trapdoors, we point out that trapdoor hash functions and *randomized trapdoor hash functions* are an interesting tool in themselves, allowing efficiency and security improvements on many signature problems [156, 244, 211, 143].

3.1.2 Discrete logarithm-based hash functions

The discrete logarithm problem in a group G is the following:

Problem 3.2 (Discrete Logarithm Problem in a cyclic group G or DLP) *Given a generator g of G and a uniformly randomly chosen element h of G, find an integer e such that $h = g^e$. (We will write $e = \log_g h$.)*

Generic methods solving the discrete logarithm problem (BSGS, Pollard, Pohlig-Hellman) run in exponential time. The hardness of DLP highly depends on the group G but not on the choice of the generator; it is believed to be hard in the multiplicative group of finite fields of large characteristic p with $p - 1 = 2q$ and q prime, in large extension fields of $\mathbb{F}_2$, and in elliptic curve and some hyperelliptic curve Jacobian groups.

Although the hardness of the factorization problem and the discrete logarithm problem in finite fields cannot be strictly related, the same ideas lead to similar algorithms to solve both problems. In particular, the best discrete logarithms algorithms for finite fields today are index-calculus algorithms and number and functional field sieves that all have a subexponential complexity. Computing discrete logarithms in prime fields is usually considered a slightly harder problem than factoring [197]. Today, primes p of 1350 bits and binary fields with n about 2700 are considered to be safe [112].

In the case of elliptic curve discrete logarithm, no improvement on generic methods is known in general so elliptic curves defined over prime fields of 145 bits are considered to be safe. Discrete logarithms for integer multiplication modulo composite numbers $n = pq$ where the factorization is not given to the solver algorithm is at least as hard as the corresponding factorization problem.

The discrete logarithm problem may be generalized to the *representation problem* [54], which is hard in an cyclic group if and only if the corresponding DLP is hard.

Problem 3.3 (Representation Problem in a cyclic group G.) *Given k random elements g_i of a cyclic group G and the order q of the group, find a* representation of identity*, that is a tuple $(e_1, ...e_k) \in (\mathbb{Z}/q\mathbb{Z})^k \setminus \{(0, ...0)\}$ such that*

$$\prod_{i=1}^{k} g_i^{e_i} = 1.$$

Theorem 3.2 *In cyclic groups, the representation problem is hard if and only if the DLP is hard.*

PROOF:

- ⇒ Suppose there exists a PPT algorithm A solving the discrete logarithm problem. We construct an algorithm B as follows: on input k elements g_i, B first picks a generator g of G and uses A k times to get integers d_i such that $g_i = g^{d_i}$, then B solves the linear diophantine equation $\sum e_i d_i = 0 \bmod q$. Algorithm B then returns the tuple $(e_1 ... e_k)$ is a solution to the representation problem.

- ⇐ Suppose there exists a PPT algorithm A solving the representation problem. We construct an algorithm B as follows: on input g and h, B picks $k-1$ random values d_i and uses A on inputs h and g^{d_i} for each i. If $(e_h, e_1, ... e_{k-1})$ is the solution given by A, then B returns $e = -e_h^{-1} \sum d_i e_i \bmod q$ which satisfies $g^e = h$.

□

Damgård [82] has given a variant of Goldwasser et al.'s construction [120, 121] which leads to a claw-free function for groups G in which the discrete logarithm problem is hard. A generator g and a set of elements $a_i \in G$ are chosen, then each permutation f_i is defined as $f_i(x) = a_i g^x$. This variant is very inefficient as a hash computation requires one modular exponentiation per bit of message. The security argument for this construction is that any claw $f_i(x) = f_{i'}(x')$ would reveal the discrete logarithm of $a_i a_{i'}^{-1}$ in base g.

Chaum et al. [71] have proposed a fixed-length hash function secure if the DLP is hard in the group $\mathbb{F}_p^*$ where p and $q := (p-1)/2$ are large primes. The function is parameterized by k random elements $g_i \in \mathbb{F}_p^*$ of order q. The message is first encoded into an element $(m_1, ... m_k)$ of $\mathbb{Z}_q^n$ then the hash value $\prod_{i=1}^k g_i^{m_i} \bmod p$ is returned. The computation requires roughly 1.5 modular multiplication per message bit. The security argument is as follows: finding collisions is clearly as hard as solving the representation problem in the subgroup of index 2 of $\mathbb{F}_p^*$, which as we saw is as hard as DLP in that group.

3.1.3 Knapsack-based hash functions

Another important class of problems in cryptography are knapsack problems. The name "knapsack" comes from the following analogy: a trekker wants to fill his knapsack with the first rate of essentials according to their respective importance (*values*) and weights (*costs*). A natural question arising is whether a certain level of necessity can be reached at a given weight.

Problem 3.4 (Knapsack Problem) *Given n values $v_1, v_2, ..., v_n$ and n weights $w_1, w_2, ..., w_n$, a desired value V and a maximal weight W, decide whether there exists a subset $I \subset \{1, 2, ..., n\}$ such that $\sum_{i \in I} v_i \geq V$ and $\sum_{i \in I} w_i \leq W$.*

The knapsack problem is NP-complete, that is it belongs to the Complexity Theory class NP (the class of decisional problems with a PPT proof when the answer is "yes") and it is NP-hard, which means that if it could be solved *in the worst case* by a PPT algorithm then so could any problem in the class NP. It is currently unknown whether NP-complete problems can be solved in polynomial time, a question known as the $P = NP$ problem. However, as long as no evidence exists that $P = NP$, NP-complete problems may be considered very hard to solve at least in the worst case.

The standard formulation of the knapsack problem is decisional. Its computational version, namely computing the set I if it exists, is also NP-hard. Indeed, given an algorithm A that solves the decisional version, we may construct an algorithm B that solves the computational problem as follows. Algorithm B first forwards its input to A and receives an answer "no" or "yes". If A returns "no" B is done, otherwise it executes A again after removing n from I, that is on the $n-1$ values $v_1, v_2, ..., v_{n-1}$, the $n-1$ weights $w_1, w_2, ..., w_{n-1}$, the desired value V and the maximal weight W. If A returns "no" then n belongs to I so B set n back to I and tries removing $n-1$, otherwise it just tries removing $n-1$. Following this way B will finally solve the computational version in at most n calls to the decisional solver.

When the values and the weights are equal the problem is called *subset sum problem*; it is also NP-complete. In cryptography, a special case of the computational subset sum problem is considered.

Problem 3.5 *Given a set of n ℓ-bit integers $\{a_1, a_2, ...a_n\}$ where ℓ is a polynomial function of n, and a ℓ'-bit integer S where $\ell' \approx \ell + \log_2 n$, find a vector x of components x_i equal to 0 or 1 such that $\sum a_i x_i = S$.*

Equivalently, we may replace $\sum_i a_i x_i = S$ by $\sum_i a_i x_i = S \bmod 2^n$ in this problem: any PPT algorithm able to solve one equation can be used to solve the other equation.

Knapsack-based cryptosystems lead to very efficient and parallelizable schemes but their security is questionable despite of the NP-completeness of the knapsack problem. For cryptography, NP-hardness is irrelevant in two senses. First, the property is *asymptotic*, and knapsack problems might very well be easy for parameters of practical sizes and become difficult only for very large parameters. Second, the property is *worst-case*, meaning that NP-complete problems may be hard only for some very specific parameters and easy in average, while hardness on average would be required for cryptography.

The hardness of Problem 3.5 highly depends on how the parameters n and ℓ relate. For $\ell(n) > n^2$ and $\ell(n) = O(\log n)$ the problem is easy in average [157, 57]. For large n, Wagner's generalized birthday attack [263] can be used. For $\ell = n$ there is no algorithm running faster than in time $O(2^{n/2})$. When $\ell > 1.0629n$, the knapsack problem reduces to the problem of finding the shortest vector in a lattice (SVP, or Shortest Vector Problem) [80, 146] that is to find $\arg\min_{v\in\mathcal{L}} ||v||$ for a lattice $\mathcal{L} = \{\sum_{i=1}^{n} x_i v_i | x_i \in \mathbb{Z}\}$ defined by a set of vectors $v_i \in \mathbb{Z}^n$.

The connection between knapsack and lattice problems can be intuited as follows: let W be a large integer. If there exist $x_1, ..., x_n \in \{0, 1\}$ satisfying $\sum a_i x_i = S$ then the vector $v = (x_1, ...x_n, 0)$ is an integer linear combination of the vectors

$$\begin{aligned}
v_1 &= (1, 0, 0, ..., 0, a_1 W) \\
v_2 &= (0, 1, 0, ..., 0, a_2 W) \\
v_3 &= (0, 0, 1, ..., 0, a_3 W) \\
&\;\;...\\
v_n &= (0, 0, 0, ..., 1, a_n W) \\
v_{n+1} &= (0, 0, 0, ..., 0, -SW)
\end{aligned}$$

and it has a small norm. For $W \geq \sqrt{n}$, it can be proved that the shortest vector of the lattice generated by $v_1, ..., v_{n+1}$ provides a solution to the knapsack.

In norm 2, it is NP-hard to find even an approximation of a lattice shortest vector up to a constant factor [181]. The best PPT algorithms for this problem are variants of the LLL algorithm [165, 236]; they provide approximations up to an exponential factor in n and are practical only if $n < 100$ [238, 239, 225].

The first knapsack utilization in cryptography is the Merkle-Hellman encryption scheme [179]. In this system, an easy instance of the knapsack problem is generated (if the a_i above are such that $\sum_{j<i} a_j < a_i$ for all i then the knapsack problem is very easy) then "hidden" by a secret permutation and a modular multiplication by a secret constant. The system has been cryptanalyzed by Shamir [241] who showed how to recover the secret constant and the permutation. Later, variants of Merkle-Hellman have also been broken by extending Shamir's ideas and by *low-density* attacks for systems with large coefficients a_i [157]. Both approaches make use of lattice reduction techniques such as the LLL algorithm and improvements upon it.

Designing a hash function based on the knapsack problem seems an easier problem than designing an encryption scheme [83]. Encryption schemes have been based on weak instances of the knapsack problem to allow decryption; their security relies entirely on the way the easy instance is disguised into a generic one and no secure approach has been designed so far. For hash functions, there is no need for a decryption algorithm so the security may rely directly on the hardness of the knapsack problem.

The most notable example is Damgård compression function [83]. In this function $n = 256$ random numbers a_i of $\ell = 120$ bits are chosen and each message $m = m_1||...||m_n$ is compressed to the value

$$f(a_1||...||a_n, m) = \sum_{i=1}^{n} m_i a_i$$

where the addition is modulo 2^n. A preimage on Damgård compression function can be computed in time $O(2^{32})$ [60] but it only gives pseudo-preimages for the full hash function; another attack due to Schroeppel and Shamir computes preimages in time 2^{64} [212]. The construction was first proposed by Impagliazzo and Naor [138] who showed that the function

$$g(a_1||...||a_n, m) = a_1||...||a_n||f(a_1||...||a_n, m)$$

is one-way and a PRNG if the subset sum problem is hard on average.

The security of Damgård and Impagliazzo-Naor constructions relies on a new average hardness assumption that actually does not benefit at all from the NP-completeness of the subset sum problem. A significant improvement to knapsack-based hash functions has been given by Ajtai [19] who showed that random instances of some lattice problems are as hard as the hardest instances of some other problems. Ajtai's work and subsequent improvements [115, 58, 183, 185, 184, 202, 201, 172] have had a major influence on the SWIFFT hash function described in Section 3.3.

3.2 The Very Smooth Hash (VSH)

Hash functions based on discrete logarithm and factorization problems have been orders of magnitude slower than custom design hash functions until Contini et al. proposed VSH [76]. The function is 25 slower than SHA-1 and is collision resistant under a new assumption that is strongly related to factoring; a variant related to discrete logarithms is also proposed.

This section follows the main lines of [76]; we refer to the original paper for omitted details.

3.2.1 The VSSR assumption

An integer b is said to be B-smooth if all its prime factors are smaller than B; it is called *very smooth* when B is a polylogarithmic function of n, $B = (\log n)^c$ for some constant c. The ith prime is noted p_i, that is $p_1 = 2, p_2 = 3, ...$ and additionally $p_0 = -1$. An integer x is called a *trivial* modular square root of an integer b if b is a perfect square and $b = x^2$. Roughly speaking, the VSSR hardness assumption is that it is hard to find a non-trivial modular square root of a very smooth number.

Problem 3.6 (Very Smooth number nontrivial modular Square Root or VSSR) *Given n a product of two randomly chosen primes of about the same size, find $x \in \mathbb{Z}_n^*$ such that $x^2 = \prod_{i=1}^{\mu} p_i^{e_i} \bmod n$ for $\mu \leq (\log n)^c$ and at least one of $e_0, ...e_\mu$ is odd.*

Finding very smooth relations like $x^2 = \prod_{i=1}^{\mu} p_i^{e_i} \bmod n$ is a key step in all factorization algorithms because such relations can be combined with linear algebra to produce a pair (x, y) such that $x^2 \equiv y^2 \bmod n$, which reveals a factor of n when $x \not\equiv \pm y \bmod n$, that is with a probability 1/2. As many smooth relations are necessary for the factorization algorithms, the size of the modulus necessary to expect the hardness of VVSR is a bit larger than the size needed for the hardness of factoring [76].

3.2.2 The Very Smooth Hash algorithm

In VSH, the key generation algorithm *Gen* produces a random RSA-modulus $n = pq$ where p and q are primes of the same size. The hashing algorithm implicitly follows the structure of Merkle-Damgård. The message block length μ is defined as the largest integer such that $\prod_{i=1}^{\mu} p_i \leq n$. For a key $s = n$ and a message $m = m_0...m_{L-1}$ of $L < 2^\mu$ bits, VSH algorithm runs as follows:

1. Let $x_0 = 1$.

2. Let $N = \lceil L/\mu \rceil$ the number of blocks. Pad the last block as $m_i = 0$ for $N < i \leq N\mu$.

3. Encode the message length $L = \sum_{i=0}^{\mu} L_i 2^i$ in the block $N + 1$.

4. For $j = 0, 1, ..., N$ compute $x_{j+1} = x_j^2 \prod_{i=1}^{\mu} p_i^{m_{j\mu+i}} \bmod n$.

5. Return x_{N+1}.

The compression function implicitly used in VSH is

$$f(s,x||m) = x^2 \prod_{i=1}^{\mu} p_i^{m_i} \bmod n.$$

Although this compression function is not itself collision resistant, it satisfies weaker properties that suffices for the security of the Merkle-Damgård transform [76].

Theorem 3.3 [76] *Finding a collision in VSH is as hard as solving VSSR.*

Proof: The proof of [76] shows that two different colliding messages m and m' lead to a solution of VSSR. Let x_j, x'_j be the outputs of the successive applications of the compression function, L, L', N, N' be the lengths and number of blocks for m and m', and $m[j], m'[j]$ be the jth blocks of m and m'.

First consider the case $L = L'$. Let $t \leq N$ be the largest index such that $(x_t, m[t]) = (x'_t, m'[t])$ but $(x_j, m[j]) \neq (x'_j, m'[j])$ for $t < j \leq N+1$. Then

$$(x_t)^2 \prod_{i=1}^{\mu} p_i^{m_{t\mu+i}} \equiv (x'_t)^2 \prod_{i=1}^{\mu} p_i^{m'_{t\mu+i}} \bmod n.$$

Let $\Delta = \{i : m_{t\mu+i} \neq m'_{t\mu+i}, 1 \leq i \leq \mu\}$ and $\Delta_{10} = \{i \in \{1, ..., \mu\} : m_{t\mu+i} = 1$ and $m'_{t\mu+i} = 0\}$. We have

$$\left[(x_t/x'_t) \prod_{i \in \Delta_{10}} p_i \right]^2 \equiv \prod_{i \in \Delta} p_i \bmod n.$$

If $\delta \neq \emptyset$ this equation solves VSSR, otherwise $(x_t)^2 = (x'_t)^2 \bmod n$. If $x'_t \not\equiv \pm x_t \bmod n$ then VSSR can be solved by factoring n, otherwise $x'_t = -x_t \bmod n$ by definition of t hence $(x_{t-1}/x'_{t-1})^2$ is congruent to -1 times a very smooth number.

In the case $L \neq L'$, the equality $x_{N+1} = x'_{N+1}$ implies $(x_N/x'_N)^2 \equiv \prod_{i=1}^{\mu} p_i^{L'_i - L_i} \bmod n$. Since $|L'_i - L_i| = 1$ for at least one i, VSSR is solved as before. □

VSH is inspired by previous factorization and discrete logarithm-based hash functions like Shamir's and Chaum's ones [244, 71]. The value calculated here is $\prod_{i=1}^{\mu} p_i^{e_i} \bmod n$ where $e_i = \sum_{j=0}^{N} m_{j\mu+i} 2^{N-j}$; Shamir's function computes $g^m \bmod n$ for a randomly chosen g; Chaum uses many randomly chosen basis elements g_i but computes the multiexponentiation modulo a prime. The main contribution in VSH is the use of very small basis elements that increases the efficiency and preserves a very convincing security argument.

3.2.3 Cube-VSH, Fast-VSH and VSH-DL

Contini et al. [76] have given various variants of VSH with similar properties.

In the first variant, the squaring of the compression function is replaced by a cubing, that is $f(s, x||m) = x^3 \prod_{i=1}^{\mu} p_i^{m_i} \bmod n$. The new function is collision resistant assuming the hardness of computing a modular cube root of a very smooth cube-free integer of the form $\prod_{i=1}^{\mu} p_i^{e_i} \neq 1$ where $e_i \in \{0, 1, 2\}$ for all i, a problem related to RSA-inversion and conjectured to be hard [76].

In the second variant called *Fast VSH*, the primes p_i are replaced by small products of small primes. A larger modulus must be used but the security proof keeps unchanged and the resulting function is faster than VSH. For this variant, Saarinen [234] has shown that the hash value of a message leaks one bit of information on the message, more precisely it gives a linear equation satisfied by the bits of the messages.

In the third variant called VSH-DL, the composite modulus n is replaced by a prime p, leading to Chaum's function [71] where the random base elements are replaced by the primes p_i. The function is collision resistant under a new assumption called VSDL, Very Smooth number Discrete Log that according to [76] is related to the discrete logarithm assumption in prime fields.

Problem 3.7 (Very Smooth number Discrete Log Problem or VSDLP) *Given a random prime p such that $q := (p-1)/2$ is also prime, find integers $e_1, ...e_\mu$ such that $2^{e_1} \equiv \prod_{i=2}^{\mu} p_i^{e_i} \bmod p$ with $|e_i| < q$ for $i = 1, 2, ..., \mu$ and at least one of $e_1, e_2, ...e_\mu$ is non-zero.*

3.2.4 Pros and contras of VSH

Besides its proof of collision resistance, VSH turns out to be pretty efficient, requiring only one modular multiplication per $O(\log n)$ bits. Fast VSH is about 25 times slower than SHA-1 [76]. VSH can be used as a randomized trapdoor hash function, improving the efficiency of the verification algorithm of the Cramer-Shoup signature scheme [81] by 50% without loss of security. However, as pointed out by Saarinen [234] and the authors of VSH themselves, the function has some undesirable properties making it unsuitable as a general purpose hash function. The factorization of n constitutes a trapdoor, the hash can be inverted on small messages, it suffers from some malleability properties and truncated versions are not collision resistant.

The most apparent weakness is the trapdoor: given the factorization of n, collisions can be created by choosing m, m' such that the corresponding e_i, e_i' satisfy $e_i - e_i' = 0 \bmod \varphi(n)$. Advantages and drawbacks of trapdoors are discussed at the end of Section 3.1.1. In practice, if VSH is used with RSA signatures, it is advisable to choose two distinct moduli for the hash and the signature.

The VSH algorithm achieves mixing through reductions modulo n; for short messages (typically messages of length smaller than half of the block length) these reductions do not occur and the function becomes invertible. The attack works as follows: make a guess on the length, divide VSH result by the corresponding $\prod_{i=1}^{\mu} p_i^{L_i}$ and factorize the result. Invertibility of short messages is a concern in practice, for example in applications where passwords or keys are hashed. To protect against this attack, the authors of [76] propose to square the final result a constant number of times to ensure the appearance of reductions.

VSH suffers from malleability properties. For example, it is easy to produce two messages m, m' for which $VSH(m) = 4VSH(m') \bmod n$. More generally, Saarinen [234] has pointed out that for $m \wedge m' = 0...0$,

$$VSH(0...0)VSH(m \vee m') \equiv VSH(m)VSH(m') \bmod n$$

where $\wedge$ and $\vee$ are bitwise AND and OR. This property may introduce security threats against adaptive adversaries if the hash function is used as a MAC.

VSH can definitely not be used as a random oracle. However, its security proof for collision resistance is very convincing and the function is very efficient compared to early provable hash functions. If the trapdoor is carefully treated, it is a very good hash candidate for applications requiring only collision resistance, but it would be very dangerous using it in other applications.

3.3 The SWIFFT hash function

As discussed in Section 3.1.3, knapsack problems have appealed to Cryptographers because of their asymptotic hardness and because they potentially lead to very efficient, parallelizable schemes. However, the breaking of many knapsack-based encryption schemes and hash functions gave the knapsack a bad reputation. These cryptanalytic results were possible despite of the NP-hardness because hardness results from the Theory of Complexity are only *asymptotical* and *worst-case*. Indeed, the attacks against knapsack-based cryptographic schemes can be divided into attacks specific to particular knapsack problems used in the schemes, and general attacks successful for the parameter sizes proposed but asymptotically inefficient.

Knapsack problems have regained interest since Ajtai's hardness reduction of lattice problems from average case to worst case [19, 58]. Ajtai's result leads to the following function that is one-way if the shortest vector problem is hard to approximate in the worst case. The key generation algorithm produces $n_1, n_2, q \in \mathbb{N}$ such that $n_1 \log q < n_2 \leq \frac{q}{2n_1^4}$ and $q = O(n_1^c)$ for some constant $c > 0$, and a random matrix $M \in \mathbb{Z}_q^{n_1 \times n_2}$. The hashing algorithm, on inputs a key (n_1, n_2, q, M) and a message $m \in \{0,1\}^{n_2}$, returns the hash value $H(M,m) = Mm \bmod q$. The function has later been shown to be also collision resistant [115].

To improve the efficiency of Ajtai's function and to reduce the key size, Micciancio proposed to use special knapsack problems corresponding to cyclic lattices [182] that are lattices such that $(x_1, x_2, ..., x_n) \in \mathcal{L} \Leftrightarrow (x_n, x_1 ..., x_{n-1}) \in \mathcal{L}$. The efficiency is enhanced by using FFT algorithm [77] and the key size is reduced because only one row of M is needed to define it. Although no complexity hardness result is known on this special class of lattices, the best algorithms solving general lattice problems do not perform better on cyclic lattices. Moreover, the problem of finding the shortest vectors in cyclic lattices can be related to long-studied problems in Algebraic Number Theory [201, 173].

Micciancio's one-way function has been slightly modified by Lyubashevsky and Micciancio [170] and Peikert and Rosen [201] to lead to a collision resistant hash function under the assumption that the shortest vector problem on cyclic lattices is hard to solve even approximately up to a linear factor. Building on this previous works, Lyubashevsky et al. [172, 173] proposed to use a particular set of parameters specially adapted to current processor architectures. The resulting function called SWIFFT reaches throughputs comparable to those of SHA-256. It is used as a building block in the NIST's submission SWIFFTX [1].

3.3.1 The SWIFFT algorithm

The key generation algorithm *Gen* of SWIFFT generates n_1, n_2, p where n_1 is a positive power of 2, n_2 is a positive integer and p is a prime satisfying $p = 2tn_1 + 1$ for some positive integer t. It also generates a random matrix $A \in \{0,1\}^{n_1 \times n_2}$ and it outputs n_1, n_2, p, A. The particular values $n_1 = 64$, $n_2 = 16$ and $p = 257$ are suggested in [173] as large enough for security and particularly suited for software implementations.

The SWIFFT algorithm is a fixed-length hash function, mapping a binary matrix x of size $n_1 \times n_2 = 64 \times 16$ bits to an output in the range $\mathbb{Z}_p^{n_1}$ which has size $p^{n_1} = 257^{64} \approx 2^{512}$. The function is collision resistant assuming the worst-case hardness of a class of subset sum problems that corresponds to cyclic lattices.

For a key $A \in \{0,1\}^{n_1 \times n_2}$ and a message $m \in \{0,1\}^{n_1 \times n_2}$, let $a_1, ...a_{n_2}$ and $m_1, ...m_{n_2}$ be the columns of A and m seen as elements in the ring $R = \mathbb{Z}_p[X]/(X^{n_1}+1)$, that is, a column $a_i = (a_{i1}, ..., a_{i,n_1})^t$ is seen as the element $\sum_{i=0}^{n_1-1} a_{i+1}X^i \in R$. The output of SWIFFT algorithm is

$$mFFT^{-1}\left(\sum_{i=1}^{n_2} a_i m_i\right)$$

where $mFFT^{-1}$ is a bijection corresponding to the inverse of the modular FFT and all computations are done over R, that is modulo p and $X^{n_1}+1$.

For a vector $x = (x_1, ..., x_{n_1})^T \in \mathbb{Z}_p^{n_1}$, the modular Fourier transform of x is the bijection defined as

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n_1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega & \omega^2 & ... & \omega^{n_1-1} \\ 1 & \omega^2 & \omega^4 & ... & \omega^{2(n_1-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n_1-1} & \omega^{2(n_1-1)} & ... & \omega^{(n_1-1)^2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n_1} \end{pmatrix}$$

where ω is a $2n_1$th root of unity modulo t and the operations are done modulo p. This matrix-product computation can be done in time $O(n_1 \log n_1)$ using the modular FFT (mFFT) algorithm, a variant of the FFT algorithm for finite fields.

In SWIFFT algorithm, the multiplications $a_i m_i$ are computed by evaluating the polynomials a_i and m_i on the roots of $X^{n_1}+1$; as $R = \mathbb{Z}_p[X]/(X^{n_1}+1)$ this can be done efficiently with the help of the mFFT algorithm [172]. The input message m is first preprocessed by multiplying the ith row by ω^{i-1}, then one mFFT

$$(y_{1j}, ..., y_{n_1j}) = mFFT(\omega^0 x_{1j}, ..., \omega^{n_1-1}x_{n_1j})$$

is computed for each column $j = 1, ...n_2$. Finally, a linear combination involving the key elements is computed across each row $i = 1, ..., n_1$

$$z_i = a_{i1}y_{i1} + ... + a_{in_2}y_{in_2} \bmod p.$$

Performing the multiplications $a_i m_i$ would require an additional application of the mFFT algorithm but as mFFT is a bijection it is simply omitted (hence the appearance of $mFFT^{-1}$ in the definition above). The output is the vector $(z_1, ...z_{n_1}) \in \mathbb{Z}_p^{n_1}$.

Lyubashevsky et al. [173] proposed further efficiency improvements specific to the choice $n_1 = 64$, $n_2 = 16$ and $p = 257$. Taking full advantage of computer architecture and SIMD (single-instruction multiple-data) instructions, they reach throughputs comparable to SHA-256 hash function.

3.3.2 Pros and contras of SWIFFT

The SWIFFT hash function is amazingly efficient compared to other functions with "flavors of provability". The current throughput is comparable to SHA-256 and further improvements mapping the inherent parallelism in the function to computer architecture are expected.

SWIFFT has good statistical properties: it is a universal hash function, it is *regular* (in the sense that if the input is chosen uniformly at random then the output is uniformly random), and it can be used for randomness extraction (see Section 2.6.3).

SWIFFT is collision resistant under the assumption that it is hard to find relatively short nonzero vectors in n_1-dimensional cyclic lattices *in the worst case*, that is for at least one cyclic lattice. The problem is NP-hard for general lattices, and current lattice reduction algorithms seems unable to exploit the particular structure of cyclic lattices despite of investigations motivated by related problems in Algebraic Number Theory. Although current improvements on lattice reduction for cyclic lattice might be possible, the security proof of collision resistance is rather convincing. We point out however that the particular choice of parameters chosen in [173] for efficiency does not satisfy the hypothesis of Theorem 2 in [170] and hence does not benefit from the average-case to worst-case reduction.

SWIFFT hash function has some malleability issues coming from its structure. The most apparent one is that SWIFFT is a linear function:

$$SWIFFT(A, m) + SWIFFT(A, m') = SWIFFT(A, m + m').$$

The property does not contradict collision nor preimage resistance but it might be very damaging in many applications. Lyubashevsky and Micciancio have turned this apparent weakness into a strength by building an (asymptotically) efficient signature scheme based on their hash function [171].

Another important issue with SWIFFT is that many of its security properties rely on the key generation process in which trapdoors can be introduced very easily. In particular, for any two messages m and m' it is easy to find a key A such that m and m' collide, as it just amounts to solving the linear equation $\sum_i a_i(m_i - m'_i) = 0$.

SWIFFT is definitely not indistinguishable from a random oracle and its key generation process should be carefully defined to prevent trapdoor attacks. However, its security proof for collision resistance is rather convincing and it is amazingly efficient. SWIFFT has an interesting design and seems a good hash candidate for efficiency-concerned applications that only require collision resistance. Alternatively, SWIFFT is used as a building block with some additional design in the SWIFFTX proposal to NIST's call for new hash algorithms.

3.4 Block-cipher based hash functions

Block Ciphers are a fundamental primitive in modern cryptography that is both very efficient and well studied. As the Advanced Encryption Standard (AES [13]) is a very trusted algorithm, it is a natural idea to construct a hash function with a block cipher used as a black box.

This design approach was particularly meaningful at a time when no good design of hash function was known and the DES and triple DES algorithms were well-trusted [95]. It was abandoned after the adoption of SHA algorithm as a standard [14], because the algorithm was much faster than existing block cipher-based hash functions. Today, the design makes lot of sense again because the security of SHA is being questioned [266, 265] while the confidence in AES remains very high.

This section reviews the main results concerning block cipher-based hash functions. We start by describing the security model of such constructions, then we classify existing proposals in four categories as in [212], and finally we discuss advantages and drawbacks of block cipher-based hash functions.

3.4.1 The Ideal Cipher model

The security of block cipher-based hash functions is usually analyzed in the *ideal cipher model* and with respect to unbounded adversaries [178, 271, 46, 44]. The ideal cipher model has imposed in this setting because the classical model for block ciphers, the pseudorandom permutation model, is insufficient alone to construct collision resistant hash functions. Indeed, Simon [249] has given a black-box separation between one-way functions and collision-resistant hash functions, and one-way functions exist if and only if pseudorandom permutations exist [226].

Let $\kappa', \mu' \geq 1$ be numbers. A *block cipher* is a function mapping a *key* and a *message* to a *ciphertext*

$$E : \{0,1\}^{\kappa'} \times \{0,1\}^{\mu'} \to \{0,1\}^{\mu'}$$

where for each $k \in \{0,1\}^{\kappa'}$, the function

$$E_k : \{0,1\}^{\mu'} \to \{0,1\}^{\mu'} : m \to E_k(m) := E(k,m)$$

is a permutation on $\{0,1\}^{\mu'}$. Let E^{-1} be the inverse of E, that is the block cipher such that $E_k^{-1}(E_k(m)) = m$ for all $m \in \{0,1\}^{\mu'}, k \in \{0,1\}^{\kappa'}$.

Let $Block(\kappa', \mu')$ be the set of all block ciphers $E : \{0,1\}^{\kappa'} \times \{0,1\}^{\mu'} \to \{0,1\}^{\mu'}$. A security analysis in the *ideal cipher model* considers any actual block cipher like AES as a random element of $Block(\kappa', \mu')$, that is a block cipher E such that for each key $k \in \{0,1\}^{\kappa'}$ a random permutation E_k has been chosen [46].

A block cipher-based hash function is a hash function

$$H := Block(\kappa', \mu') \times \{0,1\}^* \to \{0,1\}^\lambda$$

in which the role of the hash key is played by the block cipher itself. In particular, a block cipher-based hash function is called collision resistant in the ideal cipher model if it is hard to find collisions when the block cipher is selected randomly [46].

A proof in the ideal cipher model gives good security guarantees if the underlying block cipher presents no structural weakness. In particular, the ideal cipher model provides a good abstraction of the AES algorithm today. For block ciphers like DES with complementation or other structural weaknesses, a proof in the ideal cipher model only gives security guarantees against adversaries that do not exploit these structural weaknesses. The limits of the ideal cipher model are very similar to the limits of the random oracle model (see Section 2.3.4). Indeed, the two models are equivalent as was recently shown by Coron et al. [79].

3.4.2 Main constructions

Block cipher-based hash functions follow the Merkle-Damgård paradigm that builds a hash function by iterating a compression function (see Section 2.4)

$$f : Block(\kappa', \mu') \times \{0,1\}^{\mu+\lambda} \to \{0,1\}^\lambda.$$

Following [212, 213], we define the *rate* of a block cipher-based hash function as the number of message blocks hashed per decryption or encryption operations. We also mainly follow the presentation of [213].

Single block length hash functions

When $\kappa' = \mu'$, there exist various secure *single block length* hash functions with $\mu = \lambda = \mu'$. Preneel et al. [216] have considered all constructions of type

$$f(E, h_{i-1}||m_i) := E(x_i, y_i) \oplus z_i$$

where x_i, y_i and z_i are chosen among $m_i, h_{i-1}, m_i \oplus h_{i-1}$ and a constant c. Their conclusion, confirmed by a more rigorous analysis in the ideal cipher model by Black et al. [46], is that there exist 12 secure such constructions in the sense that finding collisions and preimages require time respectively about $2^{\lambda/2}$ and 2^{λ}.

Among these 12 secure constructions, the most known are the Matyas-Meyer-Oseas scheme [175] $f(E, h_{i-1}||m_i) := E(h_{i-1}, m_i) \oplus m_i$, the Miyaguchi-Preneel scheme [188] $f(E, h_{i-1}||m_i) := E(h_{i-1}, m_i) \oplus m_i \oplus h_{i-1}$ and the Davies-Meyer scheme [85] $f(E, h_{i-1}||m_i) := E(m_i, h_{i-1}) \oplus h_{i-1}$.

Besides these 12 constructions, Black et al. have shown that 8 other schemes have the same collision resistance but a reduced security as a one-way function [46]. Further properties of the 12 basic schemes are considered in [212].

Multiple block length hash functions

Due to the birthday paradox (Section 2.5.2), the output size of collision resistant hash functions must be twice as large as the output size of most cryptographic algorithms including block ciphers. Even the AES algorithm [13] may not be used today in a single block length hash function because its output has only 128 bits. For this reason, designers have attempted to construct block cipher-based hash functions with larger output size.

No construction is known to achieve optimal collision, preimage and second preimage resistance. The most notable constructions are MDC-2, MDC-4 and Knudsen-Preneel code-based construction. The compression function of MDC-2 satisfies $\lambda = 2\mu = 2\kappa' = 2\mu'$; it is defined as

$$\text{MDC-2}(E, h^1_{i-1}||h^2_{i-1}||m_i) := h^1_i||h^2_i,$$

where

$$\begin{array}{ll} h^1_i := LT^1_i||RT^2_i & LT^1_i||RT^1_i := E(h^1_{i-1}, m_i) \oplus m_i, \\ h^2_i := LT^2_i||RT^1_i & LT^2_i||RT^2_i := E(h^2_{i-1}, m_i) \oplus m_i. \end{array}$$

The construction has appeared in [10]. Knudsen et al. [151] have recently reported preimage and second preimage attacks with complexity $2^{\lambda/2}$ and a collision attack of complexity $2^{\lambda/2}\left(\frac{\log_2(\lambda/2)}{\lambda/2}\right)$.

MDC-4 consists of two MDC-2 steps with a swapping in the middle; it has a rate 1/4. We refer to [212, 153, 158] for more details and attacks beating optimal preimage, second preimage and collision bounds. The construction of Knudsen and Preneel [153] uses linear codes in a clever way to design multiple block length hash functions with a “security proof” under suitable assumptions. However, these assumptions were partially contradicted by Watanabe [267] who presented a differential attack working for at least some parameters.

Using block ciphers with large keys

Some existing block ciphers including the AES have a mode satisfying $\kappa' > \mu'$, in which case there exist efficient and secure constructions of hash functions in the ideal cipher model.

The first and simplest scheme with rate $(\kappa' - \mu')/\mu'$ was proposed by Merkle: [177]

$$f(E, h_{i-1}||m_i) := E(m_i||h_{i-1}, c)$$

where c is a constant string. If $\kappa' = 2\mu'$ the scheme has rate 1; it has optimal preimage and collision resistance in the ideal cipher model, but in practice the collision resistance will highly depend on the key scheduling of the block cipher used [212].

Other schemes have also been designed by Lai and Massey and by Hirose; we refer to [158, 212, 131] for further details.

MAC constructions

As already discussed in Section 2.6.1, MAC algorithms can be (heuristically) derived from hash functions through standard constructions like HMAC and NMAC. In the case of block cipher-based hash functions, other MAC constructions have also been designed. In these constructions, a block cipher is used recursively to process message blocks, with a key equal to the MAC key.

The most famous construction is the CBC-MAC [25, 26, 141, 142]

$$h_i = E(s, h_{i-1} \oplus m_i),$$

with a final output transformation g to avoid the simple forgery attack that from $E(s, m_1)$, $E(s, m_1||m_2)$ and $E(s, m_1')$ returns $E(s, m_1'||m_2') = E(s, m_1||m_2)$ for $m_2' = m_2 \oplus E(s, m_1) \oplus E(s, m_1')$. The ANSI retail MAC [26] takes for example

$$g(h_N) := E(s_1, E^{-1}(s_2, h_N)).$$

Other output transformations have been proposed, as well as other block cipher-based MACs like XOR MAC, PMAC, 3GPP-MAC and XCBC. We refer to [215] for a short description of these schemes, references to the original papers, as well as generic attacks against block cipher-based MACs.

3.4.3 Pros and contra

Building hash functions from block ciphers reduces design and evaluation effort. The understanding and the trust we currently have in many block ciphers among which the AES can be somehow transferred to the hash function. Most importantly, existing implementations of block ciphers can be reused for hash functions, and in constrained environments the same resources can be shared between encryption and hashing primitives.

On the other hand, existing block cipher-based constructions are quite slow compared to custom designed hash functions because the block cipher key is changed for each message block and the key schedule part of block ciphers is often slow. This problem is unlikely to be solved [44]. Another, important problem is that block ciphers may differ from the ideal cipher model by weaknesses not relevant for encryption but very damaging in hash constructions, like the complementation property of DES. In particular, the key schedule of block ciphers might not be as strong as the block cipher itself. Finally, patents and other legal export restrictions limit the use of some block ciphers in the constructions.

3.5 Expander hashes: pros and contras

Expander hash functions will be fully described in Chapter 4 and their security will be analyzed in subsequent chapters. In this section, we synthesize the main advantages and drawbacks of this design strategy to allow comparison with other provable hash functions.

Expander hash functions are constructed from regular graphs in a very simple way: starting from some vertex in the graph, a walk is performed that depends on the message digits, the last vertex reached being the hash value (see Figure 4.4 in Section 4.2). This clear and simple design is an advantage in itself as it facilitates the security evaluation process; it also removes the necessity for a domain-extending transform like the Merkle-Damgård transform.

Some properties of expander hash functions follow from properties of the graphs used, and other properties like collision and preimage resistances admit interesting interpretations in terms of graphs. Bounds on the uniform distribution of the outputs are straightforwardly derived from the *eigenvalues* and the *expanding properties* of the graph. The minimal "distance" between any pair of colliding messages is given by the *girth* of the graph. To a collision corresponds either a *cycle* or *two paths* in the graph with the same starting and ending vertices, and to a preimage corresponds a path between two given vertices.

Expander hashes have a *malleability* property that directly follows from the design strategy: appending one digit to any message corresponds to change its hash value by one of its neighbors in the graph. This malleability property does not contradict preimage nor collision resistance but it is undesirable in many applications of hash functions (see Chapter 8).

A *Cayley hash* is a particular kind of expander hash that uses a graph constructed from a group (see Section 4.2.2). Collision and preimage resistances of Cayley hashes admit further interpretations as the hardness of some group theoretical problems, the representation, the balance and the factorization problems. These problems are hard for generic groups and as hard as the discrete logarithm problem in Abelian groups, but their complexities are unknown for most particular groups. Table 3.1 presents graph and group interpretations of expander and Cayley hashes properties.

Table 3.1: Correspondence between hash, graph and group properties. The last column only applies to Cayley hashes.

Hash properties	**Graph properties**	**Group properties**
collision resistance	cycle/ two-paths problem	representation / balance problem
preimage resistance	path-finding problem	factorization problem
output distribution	expanding properties	Kazhdan constant
minimal collision "distance"	girth	

Among all Cayley hash proposals, the Zémor-Tillich hash function is the only one surviving today. Collisions for the first Cayley hash proposal by Zémor can be found with the Euclidean algorithm and both collisions and preimages for LPS and Morgenstern hash functions can be found efficiently (see Section 5.6.1 and Chapter 6). On the other hand, despite of some cryptanalytic results, the Zémor-Tillich hash function has remained unbroken since 1994 for well-chosen parameters, and LPS and Morgenstern hash functions can be easily repaired in an apparently safe way (see Chapter 5 and Section 6.5). The hardness of the representation problem in general, hence the security of Cayley hashes, is a big open problem left by this thesis (see Section 10.3.1).

The efficiency of expander hashes differs for each particular instance. The Zémor-Tillich hash function is the most efficient expander hash but it is still

10 to 20 times slower than SHA. LPS and Morgenstern hash functions are a bit slower than Zémor-Tillich, and Pizer hash function is still an order of magnitude slower.

Cayley hashes computation can be parallelized easily, which could benefit efficiency in many applications.

The very clear and simple design of expander hashes makes them very appealing hash functions but their collision resistance, despite its flavors of "provable security", is mainly an open problem. Expander hashes are definitely not pseudorandom functions due to the malleability property inherent to their design. All proposals so far have been much slower than SHA, but the parallelization property of Cayley hashes may compensate this issue in applications requiring fast computation. In Chapter 9, we will present a hash function based on the Zémor-Tillich hash function, that preserves its graph and group-theoretical interpretations and its parallelism but does not have its malleability properties.

3.6 Conclusion and further readings

Collision resistant hash functions may be constructed under some widely-believed number theoretical assumptions and the practical hardness of some knapsack problems, but the resulting functions are orders of magnitude slower than specially designed hash functions like the SHA algorithm.

A few new algorithms have been designed in the recent years that trade efficiency and security at a different level. Their efficiency is still not comparable to SHA but they are already much faster than previous proposals; on the other hand, their security relies on less standard assumptions that are variants or particular cases of classical assumptions.

Among these proposals, the most important are VSH [76] which is only 25 times slower than SHA-1 and whose security relies on an assumption very close to the factorization assumption, and SWIFFT [173], which is as efficient as SHA-256 and whose security relies on the hardness of a particular class of lattice problems. Similarly, the security of the SQUASH algorithm [243] as a one-way function relies on an assumption close to factorizing, but SQUASH is trivially not collision resistant.

There have also been many old and recent constructions of hash functions and MAC algorithms based on block ciphers, a well-understood and well-trusted cryptographic primitive. As Section 3.4 only sketches the main constructions, we refer to [212, 46, 215] for further descriptions and references. Block cipher-based hash functions tend to be slow due to the key schedule but in some cases they are provably secure assuming that the block ciphers have “no structural weakness”.

We stress once again that the “provable” hash functions discussed here are only provable in the sense of collision resistance. Setting apart block cipher-based hash functions, the algebraic structure of all the functions described in this chapter induces non random behaviors that impede their use as general purpose hash functions. Block cipher-based hash functions do not suffer from this problem but may have other problems specific to their block structure.

In the second part of this thesis, we describe and analyze expander hash functions. Like VSH or SWIFFT, these functions are collision resistant under hardness assumptions on partially new mathematical problems, but like these functions they are not general purpose hash functions due to malleability issues. Later in the third part, we will modify the Zémor-Tillich hash function into ZesT to heuristically remove all its malleability properties.

Part II

Expander Hashes

Chapter 4

From expander graphs to expander hashes

Expander graphs have become a fundamental tool in computer science and applied mathematics. They have found applications in communication networks, error correcting codes, pseudorandomness theory and in the study of the convergence of Monte Carlo algorithms. The theory of expander graphs is very rich and beautiful; it has geometrical, combinatorial, algebraical and probabilistic interpretations. Although each of the many applications has preferred its own interpretation and definition of expansion, strong connections exist between the various definitions and all now belong to the theory of expander graphs.

An expander graph is a highly connected graph that has only a few edges per vertex. If the graph models a social network (each vertex modeling a person and each edge a friendship), a high expansion implies that *rumors will spread out very fast* in the network. In particular, random walks in regular expander graphs lead very quickly to a uniform distribution on the vertices.

The exact amount of expansion in a graph is defined by its vertex or edge expansion, by the spectral gap or by the second eigenvalue of its adjacency matrix, and in the special case of Cayley graphs by its Kazhdan constant. For very large classes of graphs, expansion in one sense is equivalent to expansion in another sense.

The idea of building hash functions from expander graphs goes back to Zémor [274, 275] and Tillich and Zémor [260, 258], and was independently rediscovered by Charles et al. [68] more than ten years later. In this thesis, we will call *expander hash functions* or simply *expander hashes* the hash functions constructed following this design, and *Cayley hash functions* or simply *Cayley hashes* the expander hashes constructed with Cayley graphs.

Some properties of expander hashes can be naturally interpreted as graph theoretical properties. The expanding property of the graph implies that the hash values of sufficiently long messages are close to uniformly distributed. Collisions and preimages correspond to *cycles* and *paths* in the graph and the *girth* of the graph gives a bound on the minimal "distance" between any pair of colliding messages.

All the graphs that have been used for expander hashes have a strong algebraic structure; their collision and preimage resistances are therefore related to (more or less new) mathematical problems. Pizer hashes are collision resistant if some problems on isogenies of supersingular elliptic curves are hard. All the other expander hash proposals are Cayley hashes, in which case collision and preimage resistances relate to the balance, representation and factorization problems.

In this chapter, we introduce expander hash functions. Section 4.1 gives basic graph definitions and fundamental results on random walks on graphs. Section 4.2 gives the expander hash construction and general properties, Section 4.3 reports all explicit constructions of expander hashes so far and Section 4.4 points out that a few older schemes can also be seen as expander hashes.

The systematic study of expander and Cayley hash properties (building upon the work of Zémor [274, 275], Tillich and Zémor [260, 258] and Charles et al. [258]) is one of the contributions of this thesis. Other contributions presented in this chapter are the introduction of the Morgenstern hash function and a method to avoid most multiplications in the computation of the LPS hash function. The method also applies to the Morgenstern hash function and to the modified versions of both functions that will be suggested in Section 6.5. Finally, the connection between older schemes and expander hash functions was worth pointing out.

4.1 Expander graphs

In this section, we give the main definitions and results concerning expander graphs. Section 4.1.1 introduces basic graph notations, properties and definitions; Section 4.1.2 gives definitions of expansion; Section 4.1.3 describes random walks in expander graphs and Section 4.1.4 discusses Cayley graphs. We focus on the expander graph properties that are relevant for expander hashes; the results are extracted for undirected graphs from the excellent survey of Hoory et al. [133] and existing extensions to directed graphs are discussed. The reader interested in the many aspects of expander graphs that we do not consider here is referred to [133].

4.1.1 Basic definitions and notations

A *graph* $\mathcal{G}$ is a couple of sets (V, E), where V is called the *vertex set* of $\mathcal{G}$ and $E \subset V \times V$ is called the *edge set* of $\mathcal{G}$. A *subgraph* $\mathcal{G}'$ of $\mathcal{G}$ is a graph whose vertex and edge sets are subsets of the vertex and edge sets of $\mathcal{G}$. Any edge $e = (v_1, v_2) \in E$ has two *endpoints*, a *starting point* v_1 and an *ending point* v_2. A *loop* is an edge (v_1, v_2) such that $v_1 = v_2$. An edge $e = (v_1, v_2)$ and a vertex v are said to be *incident* if either $v = v_1$ or $v = v_2$. The in- and out-degrees of a vertex v are the number of edges of which v is respectively the ending and starting point; these are noted $\deg^-(v)$ and $\deg^+(v)$. A vertex v_1 is *adjacent* to another vertex v_2 if (v_1, v_2) belongs to E; this is indicated by $v_1 \to v_2$. Two vertices $v_1, v_2 \in V$ are *neighbors* in the graph if at least one of (v_1, v_2) or (v_2, v_1) belongs to E; this is denoted by $v_1 \sim v_2$. An edge $e_1 = (v_{11}, v_{12})$ is *adjacent* to another edge $e_2 = (v_{21}, v_{22})$ if $v_{12} = v_{21}$.

A graph $\mathcal{G} = (V, E)$ is said to be *undirected* if for each edge (v_1, v_2) belonging to E, the edge (v_2, v_1) also belongs to E. A graph that is not undirected is called a *directed* graph. In undirected graphs the in- and -out degrees of any vertex v are equal; they are simply called the *degree* of v and noted $\deg(v)$. The *symmetrization* of a directed graph $\mathcal{G} = (V, E)$ is an undirected graph $\mathcal{G}^* = (V, E^*)$ which has the same vertex set as $\mathcal{G}$ and such that for any edge (v_1, v_2) belonging to E, the edges (v_1, v_2) and (v_2, v_1) both belong to E^*.

Graphs have a natural graphical representation made of (labeled) points, lines and arrows. To each vertex corresponds a point and to each edge corresponds an arrow line from the starting point to the ending point. In undirected graphs, arrows are naturally replaced by simple lines. Examples of such representations for a directed and an undirected graph are shown in Figure 4.1.

A *path* in a graph is a sequence of edges $(e_0, e_1, ..., e_{\mu-1})$ such that e_{i-1} is adjacent to e_i for $1 \leq i \leq \mu - 1$. The *length* of a path $(e_0, e_1, ..., e_{\mu-1})$ is μ. A path from a vertex v_1 to a vertex v_2 is a path $(e_0, e_1, ..., e_{\mu-1})$ such that v_1 is the starting point of e_0 and v_2 is the ending point of $e_{\mu-1}$. The *distance* from a vertex v_1 to another vertex v_2 is the length of the shortest path between v_1 and v_2.

A graph is *connected* if for any pair of vertices v_1 and v_2 there exists a path from v_1 to v_2 or from v_2 to v_1; it is *strongly connected* if for any pair of vertices v_1 and v_2 there exists a path from v_1 to v_2 and from v_2 to v_1.

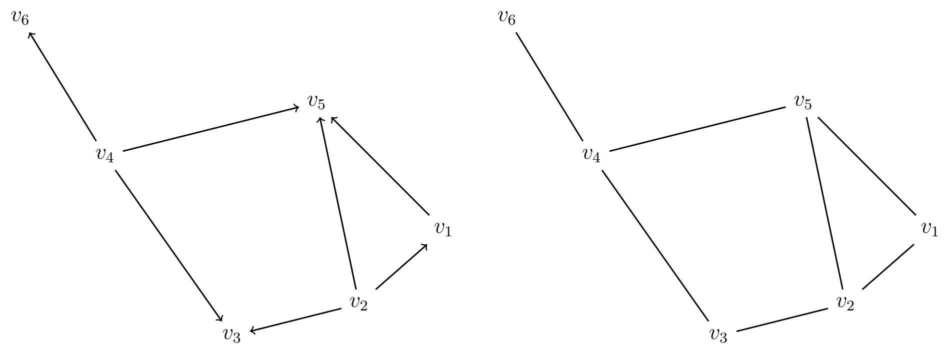

Figure 4.1: An example of drawing of a directed and an undirected graph $\mathcal{G} = (V, E)$ and $\mathcal{G} = (V, E')$ with the same vertex set $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and edge sets $E = \{(v_1, v_5), (v_2, v_1), (v_2, v_3), (v_2, v_5), (v_4, v_3), (v_4, v_5), (v_4, v_6)\}$ and $E' = \{(v_1, v_2), (v_1, v_5), (v_2, v_1), (v_2, v_3), (v_2, v_5), (v_3, v_2), (v_3, v_4), (v_4, v_3), (v_4, v_5), (v_4, v_6), (v_5, v_1), (v_5, v_2), (v_5, v_4), (v_4, v_6)\}$.

The number of connected components in a graph is the minimal number of disjoint connected subgraphs that together form the whole graph. The *diameter* D of a graph is the largest distance between any two vertices of the graph. (If a graph has two vertices with no path from v_1 to v_2, we say that its diameter is ∞.)

A path $(e_0, e_1, ...e_{\mu-1})$ is a *cycle* if the endpoint of $e_{\mu-1}$ is the starting point of e_0. The *girth* g of a graph $\mathcal{G} = (V, E)$ is the largest g such that for any two vertices v_1 and v_2 of V, any pair of distinct paths joining v_1 to v_2 is such that at least one of those paths is at least g-long. For undirected graphs, the *girth* is the smallest length of any cycle in the graph.

Graphs can be represented by their *adjacency matrix* $A_{\mathcal{G}} \in \mathbb{N}^{|V| \times |V|}$ (that we write simply A when the graph is clear from the context), whose entry (i, j) is the number of edges joining v_i to v_j. For example, the adjacency matrix of the graphs represented in Figure 4.1 are respectively

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

The adjacency matrix of an undirected graph is a symmetric matrix, which means that A is equal to its transpose matrix A'. It is easy to check that for any $\mu \geq 0$, the entry (i, j) of the matrix A^μ is the number of μ-long paths from v_i to v_j. The *eigenvalues* of a graph are the eigenvalues of its adjacency matrix, that are the roots of the polynomial equation $\det(A - \lambda I) = 0$. The eigenvalues of undirected graphs are all real; they will be noted by $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_{|V|}$.

A graph is *bipartite* if its vertex set can be divided into two disjoint subsets V_1 and V_2, such that any edge connects a vertex of V_1 to a vertex of V_2 or vice-versa. For bipartite graphs, there exists an ordering of the vertices such that the adjacency matrix has the form

$$A = \begin{pmatrix} 0 & A_{12} \\ A_{21} & 0 \end{pmatrix}.$$

More generally, a graph is m-partite if its vertex set can be divided into m disjoint subsets such that no edge has its starting and ending points in the same subset.

A *k-regular* graph is a graph such that any vertex is the starting point of exactly k edges; we say that the *degree* of such a graph is k. The elements of any column of the adjacency matrix of a k-regular graph sum up to k. Any k-regular graph has an eigenvalue equal to k to which corresponds the left eigenvector $v_1 = (1, ..., 1)$. This eigenvalue is the largest eigenvalue of the graph; its multiplicity is equal to the number of connected components in the graph. Moreover, $-k$ is an eigenvalue of a k-regular graph if and only if the graph is bipartite [84]. More generally, if there are m eigenvalues λ_j of absolute value k then $\lambda_j = k\rho^j$ with $\rho = e^{2\pi i/m}$.

A *weighted graph* $(\mathcal{G}, \omega)$ is a graph $\mathcal{G} = (V, E)$ together with a *weight function* $\omega : E \to [0, 1]$ such that $\sum_{v_2 \text{ s.t. } v_1 \to v_2} \omega((v_1, v_2)) = 1$ for all v_1. By extension, the adjacency matrix of a weighted graph is a matrix $A \in [0, 1]^{|V| \times |V|}$. Its entry (i, j) is equal to $\omega((v_i, v_j))$; it is a stochastic matrix, which means that the elements of each column sum up to 1. For undirected weighted graphs, the adjacency matrix is doubly stochastic which means that the elements of each column and each row sum up to 1.

The *normalized adjacency matrix* of a k-regular graph is its adjacency matrix divided by k; it is a stochastic matrix and a doubly stochastic matrix for undirected graphs. The *Laplacian* of an undirected graph is the matrix $L := D - A$, where D is a diagonal matrix which entry (i, i) is equal to $\deg(v_i)$. The *normalized Laplacian matrix* of an undirected graph is $I - D^{-1/2} A D^{-1/2}$. For k-regular undirected graphs, the Laplacian matrix is equal to $L = kI - A$ and the normalized Laplacian matrix to $\frac{1}{k}(kI - A)$. Laplacian matrices have an eigenvalue equal to 0 to which corresponds the left eigenvector $(1, ..., 1)$.

An *automorphism* of a graph is a permutation P of the vertices that preserves the edges, that is such that (u, v) is an edge if and only if $(P(u), P(v))$ is an edge. A graph is *vertex transitive* if for any $u, v \in V$ there is some automorphism of the graph sending u to v. Vertex transitive graphs are regular.

Graph representations may be more or less "human-readable" depending on the relative positions of the vertices; for example the graph of Figure 4.2 is identical to the right-hand graph of Figure 4.1 but much less "readable". A good heuristic for obtaining a graph representation that is well-drawn in the plane is to associate planar coordinates equal to the ith entry of the second and third eigenvectors of the graph to each vertex v_i [250].

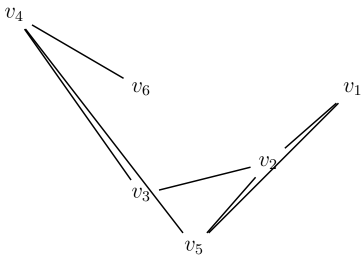

Figure 4.2: Another drawing of the undirected graph in Figure 4.1.

4.1.2 Expanding properties

The theory of expander graphs has been mainly defined for **regular undirected** graphs, in which case the analysis is facilitated by the symmetry of the adjacency matrix and in particular by the fact that all the eigenvalues are real. In this thesis, we need expanding notions for regular **directed** graphs as well, so we provide definitions that are as general as possible and discuss possible extension of classical results to directed graphs.

Let S be a subset of vertices of a graph $\mathcal{G} = (V, E)$. We write $\overline{S}$ the complement of S, that is $V \setminus S$. If S and T are two subsets of V, we write $E(S, T)$ for the set of edges of E that have a starting point in S and an ending point in T. For any subset S of V, we call *edge boundary* of S the subset $\delta(S) := E\left(S, \overline{S}\right)$. We define the expansion of a directed graph both backward and forward depending on whether we are considering the edges going in or out of the set S. The notion of a *family of k-regular expander graphs* follows naturally.

Definition 4.1 *The* forward vertex expanding constant h^+ *and the* backward vertex expanding constant h^- *of a graph $\mathcal{G} = (V, E)$ are defined as*

$$h^+(\mathcal{G}) = \min_{S \subset V, 1 \leq |S| \leq \frac{|V|}{2}} \frac{|\delta(S)|}{|S|} \quad \textit{and} \quad h^-(\mathcal{G}) = \min_{S \subset V, 1 \leq |S| \leq \frac{|V|}{2}} \frac{|\delta(\overline{S})|}{|S|}.$$

In undirected graphs as well as in directed graphs such that $\deg^+(v) = \deg^-(v)$ *at any vertex v, the forward and backward expanding constants are equal; their value is simply called the* vertex expanding constant *and it is written h.*

Definition 4.2 *A sequence of k-regular graphs $\{\mathcal{G}_n\}_{n \in \mathbb{N}}$ of size increasing with n is a* family of expander graphs *if there exists $\epsilon > 0$ such that $h(\mathcal{G}_n) \geq \epsilon$ for all n.*

The vertex expanding constants h^+ and h^- of a directed graph are related to its diameter D as follows: if $h^+, h^- \geq h$ then [275]

$$D \leq \log_{1+h} \frac{|V|}{2} + 1.$$

For undirected graphs this inequality can be improved slightly [24]. The vertex expanding constants h^+, h^- of a k-regular directed graph are related to the vertex expanding constant h of its symmetrization by [275]

$$\min(h^+, h^-) \geq h/(k+1).$$

The vertex expanding constant is sometimes called *Cheeger constant* by analogy to a similar constant in differential geometry. There exist other definitions of the boundary of a set S, for example involving neighbor vertices sets rather than edges. To each boundary definition corresponds an expanding constant definition and a definition of a family of expander graphs. Chung [74] proposed another extension of the Cheeger constant to general directed graphs. Extension of the definition of expander graph families to non regular graphs also exists but is beyond the scope of this thesis.

The edge expansion constant is the most commonly used expanding constant because of its remarkable connection with the *spectral gap* of the graph, which is the difference between its first and second largest eigenvalues (as we have already mentioned, the first eigenvalue of a k-regular graph is equal to k).

Theorem 4.1 *Let $\mathcal{G}$ be a k-regular undirected graph with spectrum $\lambda_1 = k \geq \lambda_2 \geq ... \geq \lambda_{|V|}$. Then*

$$\frac{k - \lambda_2}{2} \leq h(\mathcal{G}) \leq \sqrt{2k(k - \lambda_2)}.$$

These inequalities are often called *Cheeger inequalities*; they show that the expander constant limits the spectral gap and conversely. The result is due to Dodziuk [93] and independently to Alon-Milman [24] and Alon [21]; a proof can also be found in [133], Section 4.5. Chung [74] has given Cheeger inequalities for directed graphs; her result involves the eigenvalues of a well-defined Laplacian instead of the eigenvalues of the adjacency matrix. As far as we know, there is no result relating the (absolute values of the) eigenvalues of a directed graph to its expanding constants. A generalization would not be trivial because all current proofs exploit properties specific to symmetric matrices, in particular they use the existence of an orthonormal basis of eigenvectors.

Theorem 4.1 suggests an alternative definition of expansion based on the eigenvalues gap, which we will call *spectral expansion.* Let $\lambda_1 = k, \lambda_2, ..., \lambda_{|V|}$ be the eigenvalues of a k-regular graph and let $\lambda := \max_{i \neq 1} |\lambda_i|$.

Definition 4.3 *A sequence of k-regular graphs $\{\mathcal{G}_n\}_{n\in\mathbb{N}}$ of size increasing with n is a* family of spectral expander graphs *if there exists $\epsilon > 0$ such that $k - \lambda(\mathcal{G}_n) \geq \epsilon$ for all n.*

In the light of Cheeger's inequalities, spectral expansion implies expansion in the sense of Definition 4.2 for undirected graphs but the converse is not true in general and the two definitions cannot be related for directed graphs. The following theorem is useful to prove spectral expansion in concrete graphs.

Theorem 4.2 [75] *If $\mathcal{G}$ is a strongly connected k-regular graph and λ is its second to largest eigenvalue in absolute value, then $k > \lambda$ if and only if the greatest common divisor of all the cycle lengths of $\mathcal{G}$ is* 1.

The expansion of regular undirected graphs can not be too large because of the following lower bound.

Theorem 4.3 *For every k-regular undirected graph with $|V|$ vertices,*

$$\lambda_2 \geq 2\sqrt{k-1}\left(1 - O(1/\log^2|V|)\right).$$

In particular,

$$\lim_{|V|\to\infty} \lambda \geq 2\sqrt{k-1}.$$

The result follows from Nilli [195] and Friedman [108]; two proofs are given in Section 5.2 of [133]. The second inequality is often called the *Alon-Boppana bound* ; it does not generalize to directed graphs as shown by the "DL graphs" of Section 4.4, neither does it generalize to the Laplacian eigenvalues defined by Chung [74]. The Alon-Boppana bound motivates the following definition of a very interesting class of extremal graphs.

Definition 4.4 *A family of k-regular undirected graphs $\{\mathcal{G}_n\}$ is* Ramanujan *if $\lambda(\mathcal{G}_n) \leq 2\sqrt{k-1}$ for all n.*

Ramanujan graphs are graphs whose expansion is asymptotically maximal in the spectral sense. Alternative definitions have also been proposed that could be generalized to non regular undirected graphs, for example Definition 6.7 in [133]. The very first family of Ramanujan graphs, the LPS graph family, was discovered by Lubotzky et al. [167] and independently by Margulis. The name Ramanujan was chosen after the Ramanujan conjecture [224] for varieties over finite fields (proved by Deligne) involved in Lubotzky et al.'s proof of the eigenvalue bound. The LPS graphs are used in the LPS hash construction of Section 4.3.4.

4.1.3 Random walks on expander graphs

The previous section described some of the combinatorial and algebraic aspects of expander graphs; we now take the statistical perspective for which the spectral definition of expander graphs is the most relevant. Expander graphs share many properties with random graphs, and random graphs are indeed most likely to be expanders. The set of vertices that are reached during a random walk on an expander graph is close to what a randomly chosen set of vertices would be, and the distribution of the final vertices reached by the random walk tends very fast to the uniform distribution.

The simplest connection of expander graphs with random graphs is known as the *Expander Mixing Lemma.*

Lemma 4.1 *Let $\mathcal{G}$ be a k-regular undirected graph with $|V|$ vertices and let $\lambda = \lambda(\mathcal{G}) = \max(|\lambda_2|, |\lambda_{|V|}|)$. Then for all $S, T \subset V$:*

$$\left| E(S,T) - \frac{k|S||T|}{|V|} \right| \leq \lambda\sqrt{|S||T|}.$$

The proof of this simple result can be found in [133], Section 2.4. It uses the existence of an orthonormal basis of eigenvectors and hence it does not generalize to directed graphs. The term $E(S,T)$ is the number of edges between S and T in the graph and $\frac{k|S||T|}{|V|}$ is the expected number of edges between two randomly chosen sets of vertices; the expander mixing lemma tells us that the two terms cannot be too different in expander graphs.

We now discuss the convergence of random walks on expander graphs. Recall that a weight function $\omega : E \rightarrow [0,1]$ on the edges of the graph defines an adjacency matrix $A \in [0,1]^{|V|\times|V|}$ which is a stochastic matrix. We see *distributions on the vertices* as column vectors of length $|V|$ with entries in $[0,1]$ that sum up to 1. Given a weight function and an initial distribution π_0 on the vertices, a *random step* in the graph produces the distribution $\pi_1 := A\pi_0$. A *random walk* of length μ is obtained by iterating a random step μ times; it produces the distribution $\pi_\mu = A^\mu \pi_0$. The *standard walk* is the walk that associates the weight $\frac{1}{\deg^+(v)}$ to any edge with starting point v.

Two natural questions on random walks are whether they converge to some asymptotic distribution as their length increases, and which is the rate of the convergence. These questions have found answers in the Perron-Frobenius theory applied to stochastic matrices [160].

Stochastic matrices have an eigenvalue $\lambda_1 = 1$ to which is associated a left eigenvector $\widehat{v_1} = (1, ..., 1)$ and a positive right eigenvector v_1 called the Perron-Frobenius vector. For regular graphs and for undirected graphs, this vector is $v_1 = \frac{1}{|V|}(1, ..., 1)^t$. Increasing powers of a stochastic matrix A converge if and only if $\max_{i\neq 1}|\lambda_i(A)| < 1$. In this case, for v_1 normalized such that $||v_1||_1 = 1$, we have

$$\lim_{\mu\to\infty} A^\mu = v_1\widehat{v_1}.$$

For any initial distribution π on the vertices, the asymptotic distribution is

$$\lim_{\mu\to\infty} A^\mu \pi = v_1\widehat{v_1}\pi = v_1.$$

Random walks on spectral expander graphs always converge to an asymptotic distribution given by the Perron-Frobenius vector. The rates of convergence will be given by Equations (4.1), (4.2) and (4.3) for random walks on undirected graphs, for random walks on directed graphs and for non-backtracking random walks on undirected graphs.

For simplicity, we first describe random walks on undirected graphs. In this case, the matrix A is symmetric hence it can be written as

$$A = \sum_{i=1}^{|V|} \lambda_i v_i v_i'$$

where the λ_i are the eigenvalues of A and $\{v_i\}$ is an orthonormal basis of eigenvectors. Let again λ be the largest non-trivial absolute value of the eigenvalues, that is $\lambda = \max(|\lambda_2|, |\lambda_{|V|}|)$.

Let π be any probability distribution on V. The vector $\pi - v_1$ is orthogonal to v_1 because $v_1'(\pi - v_1) = v_1'\pi - v_1'v_1 = 0$. Let $\pi - v_1 = \sum_{i=2}^{|V|}\alpha_i v_i$ be the decomposition of $\pi - v_1$ in the orthogonal eigenvector basis. We have

$$\begin{aligned}
||A\pi - v_1||_2^2 &= ||A(\pi - v_1)||_2^2 = \left|\left|\left(\sum_{i=1}^{|V|} \lambda_i v_i v_i'\right)\left(\sum_{i=2}^{|V|} \alpha_i v_i\right)\right|\right|_2^2 \\
&= \left|\left|\sum_{i=2}^{|V|} \lambda_i \alpha_i v_i\right|\right|_2^2 = \sum_{i=2}^{|V|} |\alpha_i \lambda_i|^2 \leq \lambda^2 ||\pi - v_1||_2^2
\end{aligned}$$

and for the probability distribution $\pi = (\pi_1, ..., \pi_{|V|})'$, we have

$$\begin{aligned}
||\pi - v_1||_2^2 &= \sum_{i=1}^{|V|} \left(\pi_i - \frac{1}{|V|}\right)^2 = \sum_{i=1}^{|V|} \pi_i^2 + |V|\left(\frac{1}{|V|}\right)^2 - \frac{2}{|V|}\sum_{i=1}^{|V|} \pi_i \\
&= ||\pi||_2^2 - \frac{1}{|V|} \leq ||\pi||_1^2 - \frac{1}{|V|} < 1
\end{aligned}$$

hence

$$||A^\mu \pi - v_1||_2 \leq \lambda^\mu ||\pi - v_1||_2 < \lambda^\mu. \tag{4.1}$$

In undirected graphs, the distribution of the final vertex reached by a random walk gets closer to its asymptotic distribution by a factor at least λ at each step, resulting in an exponential decay of the distance to asymptotic distribution after only a linear number of steps.

In directed graphs, the convergence of random walk is more difficult to show because the eigenvector basis is no longer orthogonal. As we still assume that $\lambda < 1$, the random walk converges to a distribution v_1. For directed graphs, it is not possible to prove that the distribution gets closer to v_1 at any step in norm 2 but this property is true asymptotically.

Let $A = TA_JT^{-1}$ be the Gauss-Jordan decomposition of A, with $A_J = diag\{1, A_{11}, ..., A_{nn}\}$, $A_{ii} = diag\{J_{i,n_1}, ...J_{i,n_s}\}$ and

$$J_{i,n_j} = \left.\begin{pmatrix} \lambda_i & 1 & 0 & & & 0 \\ 0 & \lambda_i & 1 & 0 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & 0 & \lambda_i & 1 & 0 \\ & & & 0 & \lambda_i & 1 \\ 0 & & & & 0 & \lambda_i \end{pmatrix}\right\} n_j.$$

By exponentiating we have $A^\mu = TA_J^\mu T^{-1}$, with $A_J^\mu = diag\{1, A_{11}^\mu, ..., A_{kk}^\mu\}$, $A_{ii}^\mu = diag\{J_{i,n_1}^\mu, ...J_{i,n_s}^\mu\}$ and $(J_{i,n_j}^\mu)_{a,b} = \begin{pmatrix} \mu \\ l \end{pmatrix} \lambda_i^{p+b-a}$. For $A_1 = diag(1, 0, ..., 0)$

and $\hat{A}_J = A_J - A_1$, we get

$$\begin{aligned} ||A^\mu - TA_1T^{-1}||_2 &= ||T(A_J^\mu - A_1)T^{-1}||_2 \\ &\leq \kappa_2(T)||A_J^\mu - A_1||_2 = \kappa(T)||\hat{A}_J^\mu||_2 \end{aligned}$$

where $\kappa_2(T) = ||T||_2||T^{-1}||_2$ is the condition number of T. The matrix TA_1T^{-1} can be identified with $v_1\hat{v}_1$: the first column of T and the first row of T^{-1} are respectively the right and left eigenvectors of A associated to the eigenvalue 1. We finally get for any initial distribution π

$$\begin{aligned} ||A^\mu\pi - v_1||_2 &= ||A^\mu\pi - v_1\hat{v}_1\pi||_2 = ||A^\mu\pi - TA_1T^{-1}\pi||_2 \\ &\leq ||A^\mu - TA_1T^{-1}||_2||\pi||_2 \leq \kappa(T)||\pi||_2 \cdot ||\hat{A}_J^\mu||_2. \end{aligned}$$

By Gelfand's formula, $\lim_{\mu\to\infty} ||\hat{A}_J^\mu||_2^{1/\mu} = \rho(\hat{A}_J)$ where $\rho(\hat{A}_J) = \max\{|\lambda_i(\hat{A}_J)|\} = \max_{i\neq 1}\{|\lambda_i(A)|\} = \lambda$ is the spectral radius of $\hat{A}_J$. Therefore,

$$\lim_{\mu\to\infty} \frac{||A^\mu\pi - v_1||_2^{1/\mu}}{\lambda} \leq 1, \qquad (4.2)$$

which shows that asymptotically at least, the norm-2 distance to the asymptotic distribution decreases by a constant factor at least λ at each step. A non-asymptotic exponential decay may also be obtained if by using the χ-square metric instead of the norm-2 distance [75].

The relevant eigenvalues here are the eigenvalues of A, in particular its second to largest eigenvalue in absolute value. The eigenvalues used by Chung to generalize the Cheeger inequalities [74] are not relevant here, even if for undirected graphs the two definitions are equivalent. The eigenvalues defined by Chung are relevant for the convergence of some *lazy* walks, that are walks which with probability $\frac{1}{2}$ do not move at any neighbor.

We now turn back to undirected graphs to discuss the convergence rate of *non-backtracking* random walks that are especially relevant for the analysis of expander hashes constructed from undirected graphs. A non-backtracking walk in an undirected graph is a walk that does not backtrack, that is no edge (v_1, v_2) in any path can be directly followed by the edge (v_2, v_1). A non-backtracking random step from a vertex v_2 coming from v_1 is performed by randomly choosing a neighbor vertex of v_2 that is different from v_1. In a non-backtracking random walk from a vertex v, one neighbor of v is randomly chosen in the first step and then successive non-backtracking steps are performed on the graph. Non-backtracking random walks starting from general distributions are defined accordingly, that is by doing a weighted sum of the distributions resulting from the non-backtracking random walks starting from each vertex.

Alon et al. [22] have studied these walks in regular undirected graphs and related their convergence to the convergence of classical random walks in the same graphs. Let $P^{(\mu)}_{uv}$ be the probability that a non-backtracking walk of length μ that starts from u ends up in v; the probability that a classical walk of length μ that starts from u ends up in v is the entry (u, v) of the matrix A^μ and will be noted A^μ_{uv}. Alon et al. have shown that non-backtracking walks in regular undirected graphs converge if the corresponding standard walks converge. The *mixing rate* and the *non-backtracking mixing rate* of a random walk and non-backtracking random walk with adjacency matrix A and asymptotic distribution v_1 are [22]

$$\begin{aligned} \rho &= \limsup_{\mu\to\infty} \max_{u,v\in V} \left|A^\mu_{uv} - v_1\right|^{1/\mu}. \\ \widetilde{\rho} &= \limsup_{\mu\to\infty} \max_{u,v\in V} \left|P^{(\mu)}_{uv} - v_1\right|^{1/\mu}. \end{aligned}$$

The first rate is equal to λ because $\rho = \limsup_{\mu\to\infty} ||A^\mu - v_1\widehat{v_1}||^{1/\mu}_{max} = \limsup_{\mu\to\infty} ||A^\mu - v_1\widehat{v_1}||^{1/\mu}_2 = \lambda$ as the matrix max norm and norm 2 are equivalent. Alon et al. show that

$$\widetilde{\rho}(\mathcal{G}) = \psi\left(\frac{\lambda k}{2\sqrt{k-1}}\right)/\sqrt{k-1}, \tag{4.3}$$

with the function ψ defined as

$$\psi(x) = \begin{cases} x + \sqrt{x^2 - 1}, & \text{if } x \geq 1; \\ 1, & \text{if } 0 \leq x \leq 1. \end{cases}$$

In particular if $\lambda \geq \frac{2\sqrt{k-1}}{k}$ then

$$\frac{k}{2(k-1)} \leq \frac{\widetilde{\rho}}{\lambda} \leq 1$$

and if $\lambda < \frac{2\sqrt{k-1}}{k}$ and $d = |V|^{o(1)}$ then

$$\frac{\widetilde{\rho}}{\lambda} = \frac{k}{2(k-1)} + o(1)$$

where the $o(1)$ term tends to 0 as $|V| \to \infty$.

The convergence of random walks can be also characterized by their *mixing time*

$$\tau(\epsilon) := \min\{\mu : ||A^\mu - v_1\widehat{v_1}|| \leq \epsilon\}$$

where $\epsilon > 0$ and $||.||$ is some matrix norm. The mixing time can be related to the diameter in undirected graph and to a modified version of the diameter in directed Cayley graphs [189].

We conclude this section with another random property of random walks in expander hashes: the probability that a random walk stays in a set of vertices decreases exponentially with its length. This result can be found in Ajtai et al. [20] and Alon et al. [23]; a proof is given in [133]. The proof generalizes to regular directed graphs if the eigenvalues are replaced by singular values.

Theorem 4.4 *Let $\mathcal{G}$ be k-regular undirected graph with $|V|$ vertices and let $\lambda = \alpha k := \max(|\lambda_2(\mathcal{G})|, |\lambda_{|V|}(\mathcal{G})|)$. Let $B \subset V$ with $|B| = \beta|V|$. Then the probability that a random walk starting from a vertex uniformly chosen in B stays in B, is bounded by $(\beta + \alpha)^t$.*

4.1.4 Cayley graphs

A *Cayley graph* $\mathcal{C}_{G,S}$ is a graph constructed from a group G and a subset S of G as follows: V contains a vertex v_g associated to each element $g \in G$, and E contains the directed edge (v_{g_1}, v_{g_2}) if and only if there is an $s \in S$ such that $g_2 = g_1 s$. For a set S of size k, the Cayley graph $\mathcal{C}_{G,S}$ is a k-regular graph. If S is symmetric, that is if $s \in S$ if and only if $s^{-1} \in S$, the graph is undirected. The graph is connected (and even strongly connected) if and only if the elements of S generate the whole group. Cayley graphs are *vertex transitive* as for any $g_1, g_2 \in G$ the mapping $v_x \rightarrow v_{g_2 g_1^{-1} x}$ is a graph automorphism that sends g_1 to g_2. The elements of S will be called the *graph generators*. An example of Cayley graph is represented in Figure 4.3.

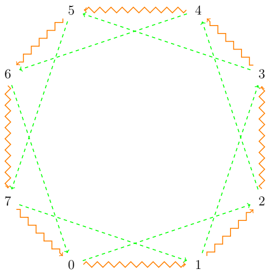

Figure 4.3: Cayley graph $\mathcal{C}_{G,S}$ where G is the additive group modulo 8 and $S = \{1, 2\}$.

The diameter of Cayley graphs of finite non-Abelian groups is often expected to be small. For finite simple groups, this follows from Babai's conjecture [30].

Conjecture 4.1 (Babai) *There exists a constant c such that for any non-abelian finite simple group G, for any symmetric set $S \subset G$ that generates G,*

$$D(\mathcal{G}_{G,S}) < (\log |G|)^c.$$

If K is a finite field, the conjecture is true for the group $PSL(2, K)$ if and only if it is true for the (non simple) group $SL(2, K)$; for $K = \mathbb{F}_p$ it was proved by Helfgott [129].

An infinite family of groups $\{G_n\}$ *can be made into a family of expanders* if there is some constant k and a generating set S_n of size k in each G_n so that the family $\{\mathcal{C}_{G_n,S_n}\}$ is a family of expanders [133]. Abelian groups and more generally *solvable* groups of bounded length cannot be made expanders with generating sets of bounded size [133, 168]. On the other hand, most simple groups can be made into families of expanders, as well as the special linear groups $SL(d, p^m)$ for any $d \geq 2$, $m \geq 1$ and prime p [149].

The study of expanding properties in Cayley graphs constructed from Abelian groups G amounts to the study of its characters, that are the homomorphisms from G to $\mathbb{C}$.

Proposition 4.1 *Let A be the normalized adjacency matrix of a Cayley graph $\mathcal{C}_{G,S}$. Let χ be a character of G. Then the vector $(\chi(g))_{g \in G}$ is an eigenvector of A with eigenvalue $\frac{1}{|S|}\sum_{s \in S} \chi(s)$.*

Proposition 4.1 is Proposition 11.7 in [133]. This approach is generalized in non-Abelian groups G by the study of the *representations* of the groups, that are homomorphisms from G to matrix groups over $\mathbb{C}$. However, these representations are often hard to analyze in practice except for generating sets S with a special structure [133].

The notion of expansion is translated in group terminology by the *Kazhdan constant.* Let the *regular representation* r of a group G be the representation that to any $g \in G$ associates a matrix of size $|G|$ which is 1 at the entries corresponding to (u, ug) for all u and 0 elsewhere. The Kazhdan constant of G and S is defined by

$$K(G, S) = \min_{v \in \mathbb{C}^{|G|}, v \perp 1} \max_{s \in S} \frac{||r(s)v - v||^2}{||v||^2}.$$

For a group G and a symmetric subset thereof S of size k, the Kazhdan constant $K(G, S)$ is related to the spectral gap of $\mathcal{C}_{G,S}$ by [133]

$$\frac{K(G, S)}{2k} \leq k - \lambda_2 \leq \frac{K(G, S)}{2}.$$

4.2 Expander hashes

We now describe the expander hash construction and its properties. Sections 4.2.1 and 4.2.2 give the construction for general and Cayley graphs. Sections 4.2.3 and 4.2.4 provide graph-theoretical and group-theoretical interpretations of collision and preimage resistances. Section 4.2.5 relates the output distribution with the expansion property, discusses the use of expander hashes as universal hash functions and points out the limits of randomness extraction with expander hashes. Finally, Sections 4.2.6 and 4.2.7 describe generic attacks against expander hashes and the malleability property that is inherent to their design.

4.2.1 General construction

An expander hash function is determined by the description of a regular expander hash, a *starting point* or a *starting edge*, and a *neighbor ordering function*. This section first gives the hash algorithm first for directed graphs then for undirected graphs, and subsequently discusses possible key generation algorithms.

Let $\mathcal{G}$ be a k-regular directed expander graph. (We assume that the graph is strictly directed which means that it has no undirected edge.) Let v_0 be a vertex in this graph that we will call the *starting point*. Let $\theta : V \times \{0, ..., k-1\} \to V$ be a *neighbor ordering function* which means that for any $v \in V$, the set $\{\theta(v, i) | i \in \{0, ..., k-1\}\}$ contains all the vertices to which v is adjacent. Given a message m, the hash algorithm decomposes this message into k-digits, that is $m = m_0 ... m_{\mu-1}$ with $m_i \in \{0, ..., k-1\}$. Then from $i = 0$ to $\mu - 1$ it successively computes $v_{i+1} := \theta(v_i, m_i)$ and it finally returns $v_{\mu-1}$. To any hash computation corresponds a walk in the graph of which the successive vertices are $v_0, v_1, ..., v_{\mu-1}$ (see Figure 4.4), the hash value being the last vertex reached by the walk. Of course, the vertices should be mapped to bitstrings in a bijective way to conform with definitions such as Definition 2.4. In the following, we abuse notations by implicitly extending all our definitions from bitstrings to vertices of a graph. Conforming with the original definitions just requires defining a bijective final map.

For k-regular undirected graphs the construction is identical except that the neighbor ordering function is adapted to explicitly forbid backtracking. Indeed, if backtracking is allowed then trivial collisions are obtained by moving from the starting point to one of its neighbors, then turning back to the starting point. Let $k' := k - 1$. The neighbor ordering function $\theta : E \times \{0, ..., k' - 1\} \to V$ now takes an edge rather than a vertex as first argument, and a symbol in $\{0, ..., k' - 1\}$ rather than in $\{0, ..., k - 1\}$

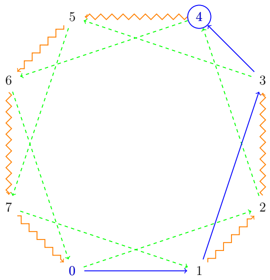

Figure 4.4: Computation of the hash value of $m = 101$ in the graph of Figure 4.3, with starting vertex 0 and a neighbor ordering function defined by $\theta(v, 0) = v + 2 \bmod 8$ and $\theta(v, 1) = v + 1 \bmod 8$.

in the second argument, and is such that for any $e = (v_1, v_2) \in E$, the set $\{\theta(e, i)|0 \leq i \leq k' - 1\} \bigcup \{v_1\}$ is the set of neighbors of v_2. Similarly, the starting point is replaced by a *starting edge* $e_0 = (v_{-1}, v_0)$ in the construction.

Given a message m, the hash algorithm decomposes this message into k'-digits, that is $m = m_0...m_{\mu-1}$ with $m_i \in \{0, ..., k' - 1\}$. Then from $i = 0$ to $\mu - 2$ it successively computes $v_{i+1} := \theta(e_i, m_i)$, $e_{i+1} := (v_i, v_{i+1})$ and it finally returns $v_{\mu-1}$. To any hash computation corresponds a walk in the graph of which the successive vertices are $v_0, v_1, ..., v_{\mu-1}$ (see Figure 4.5), the hash value being the last vertex reached by the walk.

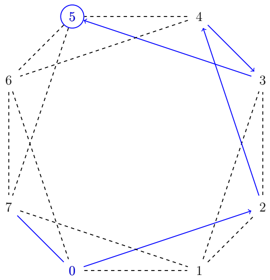

Figure 4.5: Computation of the hash value of $m = 1202$ in a 4-regular graph, with starting edge $(7, 0)$ and a neighbor ordering function such that for any $e \in E$, $\theta(e, 0) < \theta(e, 1) < \theta(e, 2)$.

These two constructions only make sense if the neighbors of any vertex in the graph may be computed efficiently. (Of course, all existing expander hash constructions satisfy this requirement.) We also remark that the output of the hash function is a vertex in the graph, which is an abstract object. To match Definition 2.4 and most applications, this output should be mapped to a bitstring in an efficient injective way. When the vertex size is not a power of two, which is the case in all expander hash proposals so far, the final mapping cannot be bijective and hence some output bits might be weaker than others. In most of this thesis, we will ignore this problem and consider the trivial generalizations of hash function definitions from bitstrings output sets to arbitrary output sets. In Chapter 9, we will come back to this issue when trying to design a practical hash function from the Zémor-Tillich hash function.

Expander hashes may be seen as the Merkle-Damgård transform of a very simple compression function. This compression function may be defined as

$$h(s, v||m_i) := \theta(v, i)$$

in the directed case and as

$$h(s, (v_1, v_2)||m_i) := (v_2, \theta((v_1, v_2), m_i))$$

in the undirected case. The comparison does not go much further because these compression functions are trivially not preimage resistant if the degree of the graph is small, which is the case in all existing proposals. The functions h above are collision resistant if there is no multiple edge, but simply because they are bijective in that case. Alternative compression functions h' may be defined as the composition of many steps h; the assumption underlying the expander hash design is that h' may become preimage and collision resistant if a sufficient number of steps are performed.

The key generation algorithm may *a priori* be defined in various ways, depending on "where the randomness is put". We stressed in Chapter 2 that some randomness must be set in the key for the definition of collision resistance to make sense. This randomness may be introduced in the choice of the parameters of the graph family, in the vertex or edge starting point and/or in the neighbor ordering function. The bibliography of expander graphs has not explicitly considered this issue so far; as the starting point or vertex and the neighbor ordering function have often been fixed in existing constructions, the randomness was implicitly put in the graph parameters.

From now on, we will assume that the neighbor ordering function is fixed for all graphs in the family, hence it may be considered as a part of the hash algorithm definition rather than as a part of the key. A collision for some neighbor ordering function can be easily translated into a collision for another neighbor ordering function (all the remaining parameters being equal), hence varying this function does not add any security.

Introducing randomness in the starting point or edge is a natural idea and it facilitates the proofs that the output distribution is uniform. However, if the randomness is set only in the starting point, the hash function is definitely not strongly universal (see Section 4.2.5). Moreover, varying the starting point does not add any security in the particular case of Cayley hashes (see Section 4.2.3), and it introduces trapdoor attacks if some starting points are weaker than others.

Randomness must be put in the choice of the graph parameters, at least for Cayley hashes. For example, the polynomial of the Zémor-Tillich hash function or the large prime of LPS hash function have to be chosen according to some distribution on the set of "good" parameters. Additional randomness on the starting point may be useful for some applications but is not mandatory; in existing proposals the starting point has always been set constant. We will specify later for each particular function how the parameters should be chosen.

4.2.2 Cayley hashes

Cayley hashes are expander hashes constructed from expander Cayley graphs. In a Cayley graph $\mathcal{G}_{G,S}$, the vertices are identified to group elements and the edges to the elements of S. The neighbor ordering functions have efficient descriptions with the elements of S.

In directed Cayley hashes, the set S is such that for any $s \in S$, $s^{-1} \notin S$. We order the elements of S as $S = \{s_0, ..., s_{k-1}\}$. The starting point is an element $g_0 \in G$, and the neighbor ordering function takes the simple form

$$\theta : G \times \{0, ..., k-1\} \rightarrow G : (g, i) \rightarrow gs_i.$$

Given a message $m = m_0...m_{\mu-1}$ where $m_i \in \{0, ..., k-1\}$, the hash value of m is

$$H(m) = g_0 \prod_{i=0}^{\mu-1} s_{m_i} := g_0 s_{m_0} s_{m_1} ... s_{m_{\mu-1}}.$$

In undirected Cayley hashes, the set S is symmetric which means that for any s in S, s^{-1} also belongs to S. Let again $k := |S|$ and $k' := k-1$. The starting edge is an edge $(g_0 s_{-1}^{-1}, g_0)$ where $g_0 \in G$ and $s_{-1} \in S$. The neighbor ordering function $\theta : S \times \{0, ..., k'-1\} \rightarrow S$ now maps an element of S and a k'-digit to an element of S; it is such that for any $s \in S$,

$$\{\theta(s, i) | 0 \leq i \leq k'-1\} \cup \{s^{-1}\} = S.$$

Given a message $m = m_0 ... m_{\mu-1}$ where $m_i \in \{0, ..., k'-1\}$, the hash algorithm successively computes $s_i = \theta(s_{i-1}, m_i)$ and $v_i = v_{i-1}s_i$ for i from 0 to $\mu - 1$. The hash value of m is

$$H(m) = g_0 \prod_{i=0}^{\mu-1} s_i := g_0 s_0 s_1 ... s_{\mu-1}.$$

The key of a Cayley hash is composed of the description of a group G, a subset S of this group, an initial group element (plus an initial subset element for undirected hashes) and a neighbor ordering function. Most often the randomness of the key will be put in the parameters defining G and all the other parameters will be considered as part of the hash algorithm.

4.2.3 Paths and cycles-finding problems

Collision and preimage resistance of expander hashes have natural graph interpretations. Finding a collision is finding *two paths* starting at the origin and ending at the same vertex. If the graph is undirected, this is just as hard as finding a *cycle* through the origin. Finding a preimage is finding *a path* starting at the origin and ending at some given vertex. (See Figure 4.6.)

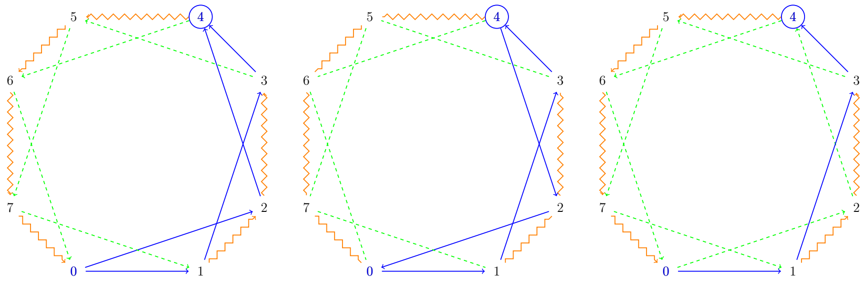

Figure 4.6: Finding a collision is finding two paths from the origin to the same vertex, hence a cycle for undirected graphs. Finding a preimage is finding a path between two vertices.

Problem 4.1 (Constrained two-paths problem) *Given a (randomly selected) starting point v_0 in a (randomly selected) graph $\mathcal{G}$, find two paths in $\mathcal{G}$ of length at most L that start in v_0 and end at the same vertex.*

Problem 4.2 (Constrained cycle problem) *Given a (randomly selected) starting point v_0 in a (randomly selected) graph $\mathcal{G}$, find a cycle in $\mathcal{G}$ of length at most L that goes through v_0.*

Problem 4.3 (Path problem) *Given a (randomly selected) starting point v_0 and an ending point v in a (randomly selected) graph $\mathcal{G}$, find a path in $\mathcal{G}$ of length at most L that starts in v_0 and ends in v.*

Problem 4.4 (Two-path problem) *Given a (randomly selected) graph $\mathcal{G}$, find two paths in $\mathcal{G}$ of length at most L that start and end at the same vertices.*

Problem 4.5 (Cycle problem) *Given a (randomly selected) graph $\mathcal{G}$, find a cycle in $\mathcal{G}$ of length at most L.*

Problems 4.1 to 4.3 are relevant for the collision and preimage resistances of expander hashes. The hardness assumptions related to these problems is that they are hard when the graph and/or the starting point are randomly selected. For the notion of hardness to make sense the graph $\mathcal{G}$ must be taken from a family of graph $\{\mathcal{G}_n\}$ with increased size. The security parameter related to these problems may be taken as the logarithm of the number of vertices in the graphs. Of course, the hardness of Problems 4.1 to 4.5 depends on each particular family of graphs. In this thesis, we will discuss families for which they have been invalidated and others for which they still seem plausible.

The problems are parameterized by the maximal length L allowed for the paths. In the case of Cayley hashes, the path corresponding to the factorization $s^{\mathrm{ord}(s)} = 1$ provides a trivial collision with the void message for any $s \in S$, but is useless in practice if the elements of S have very large order. Problems 4.4 and 4.5 may be easier than Problems 4.1 and 4.2 in general but they are equivalent in the important case of Cayley hashes. Indeed, $\prod_{i=0}^{\mu-1} s_i = \prod_{i=0}^{\mu'-1} s'_i$ if and only if $g_0 \prod_{i=0}^{\mu-1} s_i = g_0 \prod_{i=0}^{\mu'-1} s'_i$ for any $g_0 \in G$.

The girth of the graph can be interpreted as the smallest "distance" between any two colliding messages, in the sense that if $m = m_1||m_2||m_3$ and $m' = m_1||m'_2||m_3$ hash to the same value (where m_1, m_2, m'_2, m_3 are digit strings) then the length of at least one of m_2 or m'_2 is at least equal to the girth. A graph with a small girth may lead to an insecure hash function, especially if the attacker may choose the initial point of the hash computation or if the graph is vertex transitive (in particular if it is a Cayley graph). In a vertex transitive graph with small girth, there exist short collisions starting from any vertex, hence whatever the starting point is, a short collision can be found by exhaustive search. For graphs that are not vertex transitive, even cycles or "two-paths" of length 1 may be acceptable if their vertices are hard to reach from a given (possibly, a randomly chosen) starting point (Figure 4.7). However, in this case the key generation algorithm must be trusted as trapdoor attacks may be mounted by the person who chooses the initial vertex (see Section 4.2.6).

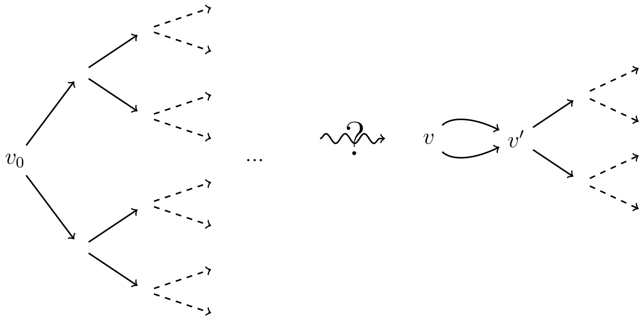

Figure 4.7: If the initial vertex v_0 is randomly chosen, a graph with small girth may still produce a safe hash. If the attacker is allowed to choose the initial point, the hash becomes insecure as he can choose $v_0 = v$.

4.2.4 Balance, representation and factorization problems

Collision and preimage resistances of Cayley hashes have further interpretations as group-theoretical problems. For a product of group elements $g_0 g_1 ... g_{\mu-1}$, let μ be the *length* of this product. A product $g_0 g_1 ... g_{\mu-1}$ is said to be a *reduced product* if $g_i g_{i+1} \neq 1$ for $0 \leq i \leq \mu - 2$.

Problem 4.6 (Representation problem) *Given a group G and a subset S thereof, find a reduced product of subset elements of length at most L that is equal to the unit element of the group, that is*

$$\prod_{0 \leq i < \mu} s_i = 1$$

with $s_i \in S$, $s_i s_{i+1} \neq 1$ and $\mu \leq L$.

Problem 4.7 (Balance problem) *Given a group G and a subset S thereof, find two reduced products of subset elements of lengths at most L that are equal, that is*

$$\prod_{0 \leq i < \mu} s_i = \prod_{0 \leq i < \mu'} s'_i$$

with $s_i, s'_i \in S$, $s_i s_{i+1}, s'_i s'_{i+1} \neq 1$ and $\mu, \mu' \leq L$.

Problem 4.8 (Factorization problem) *Given a group G, a subset S thereof and a group element g, find a reduced product of subset elements of length at most L that is equal to g, that is*

$$\prod_{0 \leq i < \mu} s_i = g$$

with $s_i \in S$, $s_i s_{i+1} \neq 1$ and $\mu \leq L$.

Problems 4.6, 4.7 and 4.8 are the group-theoretical equivalent of Problems 4.5, 4.4 and 4.3 for Cayley hashes. The problems are potentially hard only if L is *not too large* (polynomial in $\log |G|$). A trivial solution to Problem 4.6 is $s^{\mathrm{ord}(s)} = 1$ for any $s \in S$ but for large groups and well-chosen S, the length of this factorization is far too large to correspond to a practical message. Problem 4.6 is harder than Problem 4.7 in general but is equivalent if the set S is symmetric, that is for undirected Cayley hashes. Problem 4.6 is a particular instance of Problem 4.8 with $g = 1$.

Representation problems have been long studied in Spectral Graph Theory of Cayley graphs [133]; balance problems have been introduced in [38] for Abelian groups and factorization problems are well-studied in other settings. Finding *the shortest* representation and factorization are hard problems for generic groups [144, 96]. The representation problem is as hard as the discrete logarithm problem in Abelian groups [38]. For special linear groups, it has been solved for a few particular cases corresponding to Cayley hashes proposals (see Section 5.6.1 and Chapter 6) but it might still be hard in general (see Chapter 5).

Factorization problems arise naturally when trying to show that particular groups can be made into expanders or to prove Babai's conjecture (see Section 4.1.4 and [133]), two research topics that have been actively considered in the last two decades. Typically, these problems only consider symmetric sets, but we expect that any successful attack against generic undirected Cayley hashes would likely generalize to directed Cayley hashes.

It has recently been proved that special linear groups over finite fields can be made into expanders [149] and that Babai's conjecture is true for $PSL(2, \mathbb{F}_p)$ [129]. These results imply that Problem 4.8 has solutions for the corresponding groups when L is a polynomial function of $\log |G|$. However, as the proofs of these results do not produce explicit factorizations of the group elements, they are still far from solving Problems 4.6 to 4.8. To threaten the collision and preimage security of Cayley hashes in general, the expander theory of Cayley graphs and the proof techniques for Babai's conjecture would require significant further progress.

4.2.5 Output distribution and randomness extraction

We now consider statistical properties of expander hashes. The random walk theorems seen in Section 4.1.3 may be used to prove that the outputs of expander hashes are nearly uniformly distributed for any key and for sufficiently long messages. These theorems and the common use of expander graphs for randomness amplification [133] suggest that expander hashes may be used to smooth probability distributions like universal hash functions in the left-over hash lemma (see Section 2.6.3 and Lemma 2.1). However, we show that expander hashes are not universal hash functions in general and that in some cases they may remove all the entropy contained in particular messages. Expander hashes still appear promising for universal hashing and entropy extraction, but at least some further work and an adaptation of the definitions would be needed in order to derive (good proofs of) these properties.

The outputs of expander hashes are well-distributed. As shown in Section 4.1.3, a random walk in a graph converges when the largest absolute value λ of the non-trivial eigenvalues of the normalized adjacency matrix of the graph is strictly smaller than 1, hence by Theorem 4.2 if the graph is strongly connected and if the greatest common multiple of all its cycle lengths is one. Under this condition, the distribution of the expander hash values of uniformly randomly chosen messages converges to uniformity as the lengths of the messages increase. The rates of convergence are given by Equation (4.2) for expander hashes constructed from directed graphs and by Equation (4.3) for expander hashes constructed from undirected graphs.

Expander hash functions are not universal hash functions nor randomness extractors in general. Let us first assume that the graph is fixed, hence the randomness in the key is put only in the selection of an initial vertex or edge. In Cayley hashes, if $s_{m_0}...s_{m_{\mu-1}} = s_{m'_0}...s_{m'_{\mu'-1}}$ then $g_0 s_{m_0}...s_{m_{\mu-1}} = g_0 s_{m_0}...s'_{m_{\mu'-1}}$ for all g_0, hence

$$\Pr_{g_0 \in G}[H(g_0,m) = H(g_0,m')] = 1$$

which shows that Cayley hashes are not universal hash functions if the key does not include the graph parameters. Similarly for general expander hashes, the probability

$$\Pr_{g_0 \in G}[H(g_0,m) = h \wedge H(g_0,m') = h']$$

will be biased in favor of closely related messages when h and h' are neighbors in the graph. Similarly, let m_0 and m_1 be two messages corresponding to paths in the graphs that both start and end at the initial point v_0. Let us consider the set of messages of the form $m = m_{e_1}||...||m_{e_E}$, for any bitstring $e_1...e_E$. This set has an entropy E but on the other hand, all the walks defined by its messages end up at the same point, therefore contradicting randomness extraction.

Despite our counterexamples, expander hash functions might be used as universal hash functions and as randomness extractors if their key generation algorithms generate random instances of the graph parameters. However, the properties will not follow from the expander hash design. Instead, they will rather depend on each particular construction and each key generation algorithm and they might be hard to prove or disprove in practice.

Even if they are not good entropy extractors from an information-theoretic point of view, expander hashes might be used as such in a computational setting, in particular if we assume collision resistance. From cycles or two-paths in the graph, it is easy to figure out message sets with strictly positive entropy that produce vertice distributions with a very weak entropy. On the other hand, it seems hard to produce such message sets without using cycles nor two-paths. As such objects are assumed to be hard to find by collision resistance, we believe that finding a meaningful extension of the notion of entropy extraction in the computational setting is an interesting open problem.

4.2.6 Generic attacks against expander and Cayley hashes

In this section, we study the performances of generic attacks against hash functions when they are applied to expander hash functions. We first discuss improvements on generic collision and preimage attacks, including the "automorphisms attack" of [68], subgroup attacks, "meet-in-the-middle" preimage attacks, improved multicollision attacks and trapdoor attacks specific to small-girth expander hashes. We then consider differential cryptanalysis. We explain that differential cryptanalysis as it is used for other hash functions is unlikely to be applicable to expander hashes when the girth is large and we argue that for Cayley hashes, differential cryptanalysis should be replaced by subgroup attacks.

Let $\mathcal{G} = (V, E)$ be a k-regular graph. The automorphism attack described by Charles et al. [68] assumes that there exists an efficiently computable automorphism of the graph such that the average distance from a vertex v to its image $f(v)$ is small. The attacks work as follows. Take a random walk ω of length μ from the initial point v_0; let v_μ be the last vertex reached by this walk. Take a random walk ω' of length μ' from v_μ; let $v_{\mu+\mu'}$ be the last vertex reached by this walk. By a brute force attack, search for two paths ω_μ from v_μ to $f(v_\mu)$ and $\omega_{\mu+\mu'}$ from $v_{\mu+\mu'}$ to $f(v_{\mu+\mu'})$. The two paths $\omega||\omega_\mu||f(\omega')$ and $\omega||\omega'||\omega_{\mu+\mu'}$ solve Problem 4.1 with large probability.

The group structure of Cayley hashes allows for subgroup attacks, a very powerful collision attack technique. Let us suppose that there exists a subgroup tower sequence $G = G_0 \supset G_1 \supset G_2 \supset ... \supset G_N = \{I\}$ such that $|G_{i-1}|/|G_i| \leq B$ for all i and some computational bound B. By successively "going from G_{i-1} to G_i" it is possible to "reach the identity" faster than by the birthday attack. This idea was already exploited by Camion against a scheme proposed by Bosset [59, 53]; we reproduce here a description of the attack that is due to Jean-Pierre Tillich.

The attack first computes two sets of size about $\sqrt{B}$ of random products of length at most μ_1 of the graph generators s_i. The length μ_1 of the products is chosen in such a way that by taking random products of length at most μ_1 in an appropriate way we roughly get $\sqrt{B}$ different (random) coset representatives.

Choosing for each left coset of G_1 a representative, each element g of the first set can be written as $g = xg_1$, where x is one of these representatives and g_1 belongs to G_1. This element is stored in a hash table of size $\sqrt{B}$ which is used to store g and its corresponding message at the address $lb_{\log_2(B)/2}(x)$ (where $lb_{\log_2(B)/2}(x)$ is the integer given by the $\log_2(B)/2$ least significant bits of x). Choosing a representative for each right coset of G_1, each element g' of the second set can be written as $g' = g_1x$, where x is this time a representative of a right coset and g_1 belongs to G_1. If an element g has been stored at the address $lb_{\log_2(B)/2}(x^{-1})$, the product $s_{0,1} := g'g$ belongs to G_1. This operation is repeated with another choice for the second set to get a second product $s_{1,1}$ of the graph generators which belongs to G_1.

This trick is iterated from $i = 1$ to N: two elements $s_{0,i}$ and $s_{1,i}$ of G_i are obtained by random products of $s_{0,i-1}$ and $s_{1,i-1}$ of length at most μ_i. In the last step the identity is produced from two elements $s_{0,N-1}, s_{1,N-1} \in G_{N-1}$. The collision size is about $2^N\Pi_{i=1}^N\mu_i$, the storage cost is of order $\sqrt{B}$ and the computational cost of this attack is about $2N\sqrt{B}$. The storage cost can be reduced drastically with a minor increase of the computational cost with distinguished point techniques. The attack is improved if some of the random jumps from one subgroup to the next one can be replaced by PPT algorithms.

Expander hashes are also vulnerable to "meet-in-the-middle" and multi-collision attacks (Section 2.5.3). As pointed out in Section 4.2.1, the expander hash design may be seen as a Merkle-Damgård transform of a very simple compression function with message blocks made of only one digit, hence collisions can be easily combined into multi-collisions. For Cayley hashes, if

$H(m_1) = H(m_1')$ and $H(m_2) = H(m_2')$ then not only

$$H(m_1||m_2) = H(m_1||m_2') = H(m_1'||m_2) = H(m_1'||m_2')$$

but also

$$H(m_2||m_1) = H(m_2||m_1') = H(m_2'||m_1) = H(m_2'||m_1').$$

Moreover, as the compression function is obviously invertible, "meet-in-the-middle" attacks compute preimages in a time equal to the square root of the output size.

When the girth is small, the collision resistance of the function is not necessarily broken as the existing small cycles may be hard to find from a given, randomly chosen starting point v_0. In this case, there exists however an increased risk of trapdoor attack if it is possible to find the starting points of short collisions. Indeed, suppose that starting from v the messages m and m' have the same hash values. An attacker who is given the ability to choose the starting point can choose $v_0 = v$. More generally, if the "compression function" is efficiently invertible (which is usually the case with expander hashes) the attacker can produce collisions of the form $(m_1||m||m_2, m_1||m'||m_2)$ by computing the hash function "backward" from v according to the digits of m_1, then choosing the last vertex reached as starting point for the hash function. This attack can be mounted for example against our vectorial and projective versions of the Zémor-Tillich hash function (Section 5.5).

Differential cryptanalysis is unlikely to work on expander hashes, especially when the girth is large. We recall from Section 2.5.4 that these attacks were applied to compression functions that are made of many rounds. As the structure of expander hashes is very different, the attack should at least be considerably adapted. In differential cryptanalysis, the attacker searches for combinations of bit flips in the message whose changes a few rounds later are compensated with a high probability. It exploits the fact that after a small number of rounds, the change induced by some bit flips remain local: it does not influence the whole state of the algorithm. In expander hashes, the whole state is updated at each bit, and two states may coincide only after considering a number of rounds equal to the girth. For large girths, this would render the differential attack unpractical.

For Cayley hashes, differential attacks are best replaced by subgroup attacks. In a sense, the goal of differential attacks is to detect an unexpected local group structure, a group of differentials that can be combined ("added") until forming a collision ("the identity"). In traditional hash functions, these small group structures are unknown and they might once be discovered and lead to very efficient differential attacks. In Cayley hashes, the group structure is well-known and its study can be managed more easily. In particular, the absence of small subgroup structures and the girth of the Cayley graph might help providing quantitative estimations of the hardness of differential cryptanalysis.

4.2.7 Malleability properties

Malleability is an inherent property of the expander hash design. Roughly, a hash function is malleable if certain modifications on a message can be related to modifications on the hash values of the message. Malleability properties have negative implications for the security of certain hash functions applications, but on the other hand for Cayley hashes they give rise to efficient parallel computation algorithms. In this section, we describe the malleability properties of expander hashes; we will further discuss their negative and positive implications in Chapter 8.

Let H be an expander hash function constructed from a k-regular graph, and let $m_0...m_{\mu-1}$ be a message decomposed into k or $k-1$ digits (depending on whether the graph used is directed or not). Then trivially, the hash values of $m_0...m_{\mu-2}$ and $m_0...m_{\mu-1}$ are neighbors in the graph. As the neighborhood relation is efficiently computable (otherwise the hash algorithm would not be efficient), this implies that one hash value can be computed from the other, a property that is called *malleability* in other cryptographic settings [94].

If H is a Cayley hash it is even more *malleable.* Let us first assume that the starting point is the identity element of the group. For any two messages $m = m_0...m_{\mu-1}$ and $m' = m'_0...m'_{\mu'-1}$, we have

$$H(m||m') = H(m) \cdot H(m')$$

where $\cdot$ represents the group operation. This malleability property also implies that if $H(m) = 1$, then $m_1||m_2$ collides with $m_1||m||m_2$ for any messages m_1, m_2. More generally, if the starting point is an arbitrary element g_0 of the group then

$$H(m||m') = H(m) \cdot g_0^{-1} \cdot H(m').$$

These observations concluded on the security aspects of expander hashes; we now turn to concrete examples.

4.3 Expander hashes proposals

This section describes the existing instances of expander hash functions. Section 4.3.1 summarizes the necessary requirements for expander hash functions and Sections 4.3.2, 4.3.3, 4.3.4, 4.3.5 and 4.3.6 introduce the Zémor, Zémor-Tillich, LPS, Morgenstern and Pizer hash functions.

4.3.1 Necessary requirements

From our study of expander graphs and expander hashes in Sections 4.1 and 4.2, it appears that the graphs used in expander hashes should at least satisfy the following requirements.

- **Large expansion:** this requirement guarantees that the hash values of relatively short messages (with respect to the output set size) are well-distributed in the output set. The relevant parameter here is the second largest eigenvalue in absolute value determining the spectral expansion of the graph (see Sections 4.1.2 and 4.1.3).

- **Short diameter:** this requirement is related to the previous one; a short diameter implies that all vertices are the output of short messages.

- **Large girth:** this requirement guarantees that no short collision exists and it bounds the "distance" between any two colliding messages. A small girth may however be acceptable if the initial vertex is chosen randomly. For Cayley hashes, a large girth is definitely required.

- **Efficiency:** computing the neighbors of any given vertex must be (very) efficient.

- **Collision, preimage and second preimage resistances:** Problems 4.1 to 4.5 must be hard.

Cayley hashes seem particularly interesting. Their definition is even clearer and simpler than general expander hashes. They offer an additional perspective and connections to long-studied group-theoretical problems. Unlike general expander hashes, they have a high symmetry which guarantees that no starting point provides weaker security than the others. Finally, their malleability property allows parallel hash computation. We note that only non-Abelian groups may be used in this construction, otherwise trivial collisions are obtained by permuting the message digits (stated otherwise, the girth of Abelian Cayley graphs is at most 2).

Ramanujan graphs are interesting at first sight but they might be easier to cryptanalyze from the point of view of collision resistance. Ramanujan graphs are well-studied mathematical objects and a lot is known about the girths and diameters of existing constructions. By definition, Ramanujan graphs families are optimal in the spectral expanding sense described in Section 4.1.2. However, the Ramanujan property seems to be very demanding; existing constructions are actually extremal graphs in many senses and have a lot of algebraic structure. This "extra" mathematical structure is exactly what has made collision and preimage attacks feasible against LPS and Morgenstern hash functions (see Chapter 6). We also remark that the optimal expansion of Ramanujan graphs is only optimal for undirected graphs and is easily beaten in directed graphs (see Section 4.4). Using Ramanujan graphs in the expander hash construction may therefore actually lead to more drawbacks than advantages.

4.3.2 Zémor's first proposal

The expander hash design goes back to Zémor [274, 275]. His first scheme was motivated by another scheme of Godlewski and Camion [113] based upon error-correcting codes. For Godlewski-Camion hash function, a minimal distance on any pair of collisions could be inferred from the distance of the code, and the first goal of Zémor in using graphs with large girths was to derive a similar bound.

Zémor's scheme is a Cayley hash built from the group $G = SL(2, \mathbb{F}_p)$ that is the set of 2×2 matrices over the field $\mathbb{F}_p$ with unitary determinant. The graph generator set is $S_1 = \{A_1, B_1\}$ where

$$A_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad B_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

The girth of the Cayley graph $\mathcal{G}_{G,S_1}$ is larger than $\log_{\phi_1} \frac{p}{2}$ where $\phi_1 = \frac{1+\sqrt{5}}{2}$ and the diameter is in $O(\log p)$ [274]. Computation of a message of length μ requires μ multiplications by A or B, and each of these multiplications requires 2 additions modulo p. The scheme is consequently reasonably efficient.

However, Zémor's hash function has been cryptanalyzed by Tillich and Zémor [260] using a *lifting* strategy: the representation problem is lifted from $SL(2, \mathbb{F}_p)$ to $SL(2, \mathbb{Z})$ where it can be easily solved with the Euclidean algorithm (see Section 5.6.1). The attack exploits the fact that A and B generate the set $SL(2, \mathbb{Z}^+)$ in $SL(2, \mathbb{Z})$.

Two other graph generator sets are proposed in [275] to avoid the aforementioned attack, the sets $S_2 = \{A_2, B_2\}$ and $S_3 = \{A_3, B_3\}$ where

$$A_2 = A_1^2 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \qquad B_2 = B_1^2 = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$
$$A_3 = A_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad B_3 = A_1 B_1 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

The girths of the Cayley graphs $\mathcal{G}_{G,S_2}$ and $\mathcal{G}_{G,S_3}$ are respectively larger than $\log_{\phi_2} \frac{p}{2}$ and $\log_{\phi_3} \frac{p}{2}$ where $\phi_2 = 1 + \sqrt{2}$ and $\phi_3 = \frac{3+\sqrt{5}}{2}$. The proof that the diameter of $\mathcal{G}_{G,S_1}$ is polynomial in $\log p$ can be adapted to $\mathcal{G}_{G,S_3}$. It seems likely that the same property holds for $\mathcal{G}_{G,S_2}$ but further techniques are needed to prove it [275].

The scheme with $\mathcal{G}_{G,S_2}$ requires 2 additions and 2 multiplications by 2 modulo p per bit of message. For $\mathcal{G}_{G,S_3}$, it requires on average 3 additions modulo p per bit of message. Under a suitable integer representation, multiplying by 2 modulo p amounts to a one-bit shift and at most one addition.

To the best of our knowledge, these schemes have not been attacked so far. Known cryptanalytic results against Zémor-Tillich (see Chapter 5) can be partially extended to them, but if p is large enough to prevent attacks of complexity $\sqrt{p}$, and if both $p-1$ and $p+1$ have large factors to prevent subgroup attacks, both schemes seem safe today.

4.3.3 Zémor-Tillich hash function

At CRYPTO'94, Tillich and Zémor [258] replaced the group $SL(2, \mathbb{F}_p)$ in Zémor's construction by the group $SL(2, \mathbb{F}_{2^n})$, that is the group of 2×2 matrices of unitary determinant in the field $K := \mathbb{F}_{2^n}$. The elements of K are identified to binary polynomials modulo an irreducible binary polynomial $P_n(X)$ of degree n, that is

$$K \approx \frac{\mathbb{F}_2[X]}{(P_n(X))}.$$

The key of this Cayley hash contains the parameter $P_n(X)$, the starting point is the identity $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and the graph generator set is $S := \{A_0, A_1\}$ where

$$A_0 = \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix} \qquad A_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}.$$

Note that the polynomial $P_n(X)$ is implicit in the definition of the matrices A_0 and A_1. The Zémor-Tillich hash value of a bitstring $m = m_0 ... m_{\mu-1}$ is

$$H_{ZT}(P_n(X), m) := A_{m_0} ... A_{m_{\mu-1}}.$$

Quisquater has observed that this can also be written in the following very compact form

$$H_{ZT}(P_n(X), m) := \prod_{i=0}^{\mu-1} \begin{pmatrix} X & 1+m_iX \\ 1 & m_i \end{pmatrix}.$$

The Zémor-Tillich hash function is even more efficient than Zémor's previous proposals and it is particularly well-suited for hardware implementations because its computation uses only arithmetic in a field of characteristic 2. As

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} A_0 = \begin{pmatrix} aX+b & a \\ cX+d & a \end{pmatrix} \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix} A_1 = \begin{pmatrix} aX+b & aX+b+a \\ cX+d & cX+d+c \end{pmatrix},$$

a multiplication by A_0 requires computing $aX + b$ and $cX + d$. Under a suitable representation, computing $aX + b$ amounts to shifting the bits of a by one bit to the left, xoring by the polynomial if the left-most bit of a is 1 and xoring by b. In ANSI C and in hardware, the first and last step may be combined. A multiplication by A_1 requires two additional xoring for computing $(aX + b) + a$ and $(cX + d) + c$. A trivial implementation of this algorithm on a 32-bit 3.20GHz Pentium 4 using the GMP C library with a random irreducible polynomial $P_n(X)$ of degree 1024 gives a throughput of 1.4Mb/s (corresponding to 18286 cycles/byte). Further details and considerable improvements on the efficiency of the Zémor-Tillich hash function will be given in Chapter 9.

Some positive results on the security of this function are given in [258] (see Section 5.2 for further details). The set of matrices generated by A_0 and A_1 is the whole set $G = SL(2, \mathbb{F}_{2^n})$. The girth of $\mathcal{C}_{G,S}$ is at least n, the degree of the polynomial used. The diameter of $\mathcal{C}_{G,S}$ is unknown but it is expected to be poly-logarithmic in $|p|$ (for undirected graphs this would follow from Babai's conjecture (Conjecture 4.1)). Convergence of random walks to the uniform distribution is guaranteed.

A few papers have reported attacks against the collision resistance of the Zémor-Tillich hash function, all of them focusing on solving the representation problem [70, 18, 111, 251, 207]. As we will show in Chapter 5, the function remains however fundamentally unbroken today as soon as the parameters are large enough and well-chosen.

4.3.4 LPS hash function

More than 15 years after Zémor's first proposal, the idea of expander hashes has been independently rediscovered by Charles, Goren and Lauter [68]. Unlike Zémor and Tillich and Zémor, Charles et al. use undirected graphs and solve the issue of trivial backtracking collisions by explicitly forbidding backtracking in their construction. Charles et al. have proposed one scheme based on Pizer's graphs that is presented in Section 4.3.6 and another one based on the Ramanujan non-bipartite LPS construction described in the celebrated paper of Lubotzky, Phillips and Sarnak [167].

The non-bipartite LPS Ramanujan graphs are Cayley graphs defined as follows. Let p and l be primes, l small and p large, both p and l equal to 1 mod 4, such that l is a quadratic residue modulo p. Let $\mathbf{i}$ be an integer such that $\mathbf{i}^2 \equiv -1 \bmod p$. The construction uses the group $G = PSL(2, \mathbb{F}_p)$ which is usually defined as the quotient group of $SL(2, \mathbb{F}_p)$ by the equivalence relation $M \sim -M$. We will prefer the following equivalent definition: $PSL(2, \mathbb{F}_p)$ is the group of 2×2 matrices over $\mathbb{F}_p$ with non-zero square determinant, modulo the equivalence relation $M_1 \sim \lambda M_2, \lambda \in \mathbb{F}_p^*$. The set S is $S = \{s_j\}_{j=0,\dots,l}$, where

$$s_j = \begin{pmatrix} \alpha_j + \mathbf{i}\beta_j & \gamma_j + \mathbf{i}\delta_j \\ -\gamma_j + \mathbf{i}\delta_j & \alpha_j - \mathbf{i}\beta_j \end{pmatrix}, \qquad j = 0, ..., l;$$

and $(\alpha_j, \beta_j, \gamma_j, \delta_j)$ are all the integer solutions of $\alpha^2 + \beta^2 + \gamma^2 + \delta^2 = l$, with $\alpha > 0$ and β, γ, δ even[1]. The Cayley graph $X_{l,p} := \mathcal{C}_{G,S}$ is undirected since S is stable under inversion. Charles et al. suggest to use $l = 5$ and p a 1024-bit number.

The choice of LPS graphs was very appealing : they are Ramanujan and they have a large girth and a small diameter [167]

$$\begin{aligned} g(X_{l,p}) &\geq 4\log_l p - \log_l 4, \\ D(X_{l,p}) &\leq 2\log_l \frac{p(p-1)(p+1)}{2} + 2\log_l 2 + 1. \end{aligned}$$

The Ramanujan property means that all the non-trivial eigenvalues of the normalized adjacency matrix satisfy $|\lambda_i| \leq \frac{2\sqrt{l}}{l+1}$ hence $\lambda \leq \frac{2\sqrt{l}}{l+1}$ and the non-backtracking mixing rate defined in Section 4.1.3 verifies $\widetilde{\rho} \leq \frac{1}{\sqrt{l}}$. The amount of mixing per bit of message is bounded independently of l as $\left(\log_2 \frac{1}{\sqrt{l}}\right) / \log_2 l = \frac{-1}{2} = \log_2 \frac{1}{\sqrt{2}}$.

A trivial implementation of this hash function would require 8 full multiplications and 4 additions modulo p per l-digit. We have observed in [206] that the cost per digit may be reduced to a few additions and multiplications by small numbers modulo p, at the cost of doubling the memory requirements and a few full multiplications in postprocessing. Indeed, using

[1] Note that in this representation, the determinant of each graph generator s_j is l which by requirement is a quadratic residue modulo p.

$\mathbf{i}^2 = -1 \bmod p$, the multiplication of a matrix

$$M = \begin{pmatrix} a_0 + a_1\mathbf{i} & b_0 + b_1\mathbf{i} \\ c_0 + c_1\mathbf{i} & d_0 + d_1\mathbf{i} \end{pmatrix}$$

by a graph generator s_j is

$$Ms_j = \left(\begin{array}{c|c} (a_0\alpha_j - a_1\beta_j - b_0\gamma_j - b_1\delta_j) & (a_0\gamma_j - a_1\delta_j + b_0\alpha_j + b_1\beta_j) \\ +\mathbf{i}(a_0\beta_j + a_1\alpha_j + b_0\delta_j - b_1\gamma_j) & +\mathbf{i}(a_0\delta_j + a_1\gamma_j - b_0\beta_j + b_1\alpha_j) \\ \hline (c_0\alpha_j - c_1\beta_j - d_0\gamma_j - d_1\delta_j) & (c_0\gamma_j - c_1\delta_j + d_0\alpha_j + d_1\beta_j) \\ +\mathbf{i}(c_0\beta_j + c_1\alpha_j + d_0\delta_j - d_1\gamma_j) & +\mathbf{i}(c_0\delta_j + c_1\gamma_j - d_0\beta_j + d_1\alpha_j) \end{array}\right).$$

As all $\alpha_j, \beta_j, \gamma_j, \delta_j$ are smaller than $\sqrt{l}$ which is small, this product can be computed with a few additions and multiplications by small numbers modulo p. In Appendix C, we give some details for $l = 5$. As $\alpha = 1$ and exactly one of β, γ, δ is ± 2, the LPS hash function can then be computed with 7.75 additions and 3.45 one-bit shifts per bit of message. A basic implementation of this algorithm on a 32-bit 3.20GHz Pentium 4 using the GMP C library, with $l = 5$ and a random prime p of 1024 bits, gives a bandwidth of 733kb/s corresponding to 34925 cycles/byte (against 83kb/s for the trivial algorithm used in [68]).

Collisions for LPS hash function have been found by Tillich and Zémor [259]. We have extended their algorithm to a preimage attack [204]. Both attacks can be defeated with a small modification in the graph generator set. All these results are presented in details in Chapter 6 of this thesis. We remark that our little efficiency trick also applies to the modified algorithm.

4.3.5 Morgenstern hash function

Morgenstern's Ramanujan graphs [190] generalize LPS graphs from an odd prime $p \equiv 1 \bmod 4$ to any q which is an even power of 2 or a power of another prime.

Arithmetic in fields of characteristic 2 is typically more efficient and easier to implement than arithmetic modulo a large prime integer. This has led us [206] to introduce the Morgenstern hash function, which uses Morgenstern graphs for small even q.

Morgenstern graphs for even q are defined as follows. Let q be a power of 2 and let $\epsilon \in \mathbb{F}_q$ such that $f(x) := x^2 + x + \epsilon$ is irreducible in $\mathbb{F}_q[x]$. Let $P_n(X) \in \mathbb{F}_q[X]$ be irreducible of even degree $n = 2d$ and let $\mathbb{F}_{q^n}$ be represented by $\mathbb{F}_q[X]/(P_n(X))$. The construction uses the group $G = PSL_2(\mathbb{F}_{q^n})$ which can be thought of as 2×2 matrices modulo the equivalence relation $M_1 \sim$

$\lambda M_2, \lambda \in \mathbb{F}_{q^n}^*$. Let $\mathbf{i} \in \mathbb{F}_{q^n}$ be a root of $f(x)$. The set S is taken to be $S = \{s_j\}_{j=0,...,q}$, where

$$s_j = \begin{pmatrix} 1 & \gamma_j + \delta_j \mathbf{i} \\ (\gamma_j + \delta_j \mathbf{i} + \delta_j)X & 1 \end{pmatrix}, \qquad j = 0, ..., q;$$

and $\gamma_j, \delta_j \in \mathbb{F}_q$ are all the $q+1$ solutions in $\mathbb{F}_q$ for $\gamma_j^2 + \gamma_j\delta_j + \delta_j^2\epsilon = 1$. The Cayley graphs $\Gamma_{q,n} = \mathcal{C}_{G,S}$ are undirected as each s_j has order 2.

Like LPS graphs, Morgenstern graphs are Ramanujan, have a large girth, a small diameter [190]

$$\begin{aligned} g(\Gamma_{q,n}) &\geq 2/3 \log_q[q^n(q^{2n}-1)], \\ D(\Gamma_{q,n}) &\leq 2\log_q[q^n(q^{2n}-1)] + 2. \end{aligned}$$

The non-backtracking mixing rate satisfies $\widetilde{\rho} \leq \frac{1}{\sqrt{q}}$; as for LPS graphs this amounts to $\frac{1}{\sqrt{2}}$ per message bit.

Like LPS hash, Morgenstern hash function may be computed with a few additions. Using $\mathbf{i}^2 + \mathbf{i} + \epsilon = 0$, the multiplication of a matrix

$$M = \begin{pmatrix} a_0 + a_1\mathbf{i} & b_0 + b_1\mathbf{i} \\ c_0 + c_1\mathbf{i} & d_0 + d_1\mathbf{i} \end{pmatrix}$$

by a graph generator s_j is

$$Ms_j = \left(\begin{array}{c|c} (a_0 + b_0\gamma_j X + b_0\delta_j X + b_1\delta_j\epsilon X) & (a_0\gamma_j + a_1\delta_j\epsilon + b_0) \\ +\mathbf{i}(a_1 + b_0\delta_j X + b_1\gamma_j X) & +\mathbf{i}(a_0\delta_j + a_1\gamma_j + a_1\delta_j + b_1) \\ \hline (c_0 + d_0\gamma_j X + d_0\delta_j X + d_1\delta_j\epsilon X) & (c_0\gamma_j + c_1\delta_j\epsilon + d_0) \\ +\mathbf{i}(c_1 + d_0\delta_j X + d_1\gamma_j X) & +\mathbf{i}(c_0\delta_j + c_1\gamma_j + c_1\delta_j + d_1) \end{array}\right)$$

where multiplications by δ_j, γ_j and ϵ are cheap as these elements belong to $\mathbb{F}_q$ and q is small. Under a suitable representation, multiplication by X amounts to a few shifts plus a modular reduction which can be performed with a few XORs. Some details are given in Appendix C for $q = 2$, in which case the computation of Morgenstern hash function only requires 4 one-bit shifts and 12.67 XORs per bit of message: this seems to be more expensive than for the LPS hash with $l = 5$, but the XOR operations here are both easier to implement and more efficient than the additions in LPS hash. A basic implementation of this algorithm on a 32-bit 3.20GHz Pentium 4 using the GMP C library, with $l = 5$ and a random irreducible polynomial $P_n(X)$ of 1024 bits, gives a bandwidth of 613kb/s corresponding to 41762 cycles/byte, slightly less efficient than LPS hash function with $l = 5$.

The collision and preimage attacks against LPS hash can be generalized to the Morgenstern hash function with a little technical work. These attacks will be developed in Chapter 6, and a variant of Morgenstern hash that is immune to these attacks will also be proposed.

4.3.6 Pizer hash function

Among all explicit proposals of expander hashes, the Pizer hash function proposed by Charles et al. is the only expander hash using a family of graphs that is not Cayley, the Ramanujan family of Pizer [210]. This hash function has appeared in [68] and was recently discussed in a *Science* article [174].

We briefly describe the Pizer graphs used by [68] (assuming knowledge of basic results on elliptic curves which are recalled in Appendix B.5). Let l be a small prime and let p be a large prime which is congruent to 1 modulo 12. The vertices of the Pizer graph $\Pi_{l,p}$ are the set V of all supersingular elliptic curves over the finite field $\mathbb{F}_{p^2}$ (up to isomorphism). This set has $\lfloor p/12 \rfloor$ elements that can be labeled by their j-invariants; we write $E(j)$ for an elliptic curve with j-invariant j. There is an edge from j_1 to j_2 if and only if there is an l-isogeny from $E(j_1)$ to $E(j_2)$. The Pizer graph $\Pi_{l,p}$ is a Ramanujan $l+1$-regular graph; if $p = 1 \bmod 12$ it is undirected and has no multiple edge [68].

Pizer's graphs have a small diameter $D(\Pi_{l,p}) \leq 2\log_l \frac{p}{12} + 2\log_l 2 + 1$ and as they are Ramanujan, their non-backtracking mixing rate $\widetilde{\rho}$ is at most $\frac{1}{\sqrt{q}}$ which amounts to $\frac{1}{\sqrt{2}}$ per bit of message. Pizer graphs do not have small girth unless additional restrictions are put on the prime p [210, 68]. This may question the practicability of the function as it is not clear that there exists an efficient algorithm generating good p values.

Collision and preimage resistance of this function are implied by (but not equivalent to) the hardness of some isogeny problems for supersingular elliptic curves that were previously studied by Galbraith [68, 109]: the best algorithm today has a time complexity $O(p \log p)$.

The computation of a Pizer hash value is less efficient than the computation of the other expander hashes we have seen so far. To any vertex j of the graph corresponds a supersingular elliptic curve $E := E(j)$ with Weierstrass equation $Y^2 = X^3 + 3kX + 2k$ where $k = \frac{j}{j-1728}$. The l-torsion of this curve can be computed (for example, using the modular polynomial) and its subgroups can be ordered according to some convention [69, 68]. To each of these $l+1$ subgroups $H_i \subset E$ corresponds a supersingular elliptic curve $E_i = E/H_i$ and an isogeny $E \to E_i$ which can be computed using Vélu's formulae [262, 68].

The computation is the fastest when $l = 2$ but it is still at least 100 times slower than the computation of a Zémor-Tillich hash value. The 2-torsion is made of three points plus the point at infinity; if the equation of E is written in the form $Y^2 = X^3 + aX + b$, the X-coordinate of these three points are the roots of $X^3 + aX + b = 0$. Moreover, except at the first step of computation, one of these roots can easily be computed from the previous step, hence determining the whole 2-torsion amounts to factor a quadratic. Once the 2-torsion is known, Vélu's formulae require a few multiplications in $\mathbb{F}_p$. Charles et al. evaluate the total cost per bit to $2\log_2 p$ field multiplications which is much more than for the Zémor-Tillich hash function. Their implementation in C on a 64-bit AMD Opteron 252 2.6GHz has a throughput of 13.1kb/s when p has 256 bits, corresponding to 1.588M cycles/byte.

4.4 Revisiting some previous schemes

The previous section has described hash functions that were explicitly constructed from expander graphs. In this section, we show that a few other hash functions can also be interpreted as expander hashes. This simple observation brings interesting connections between old and new hash function proposals and it might open new perspectives on these old hash functions.

The easier connection is made with the claw-free permutation hashing scheme of Goldwasser et al. (Section 3.1.1 and [120, 82]) which is actually an expander hash. The implicit directed graph $\mathcal{G}_{cff} = (V, E)$ is defined as follows: V is identified to the domain of the claw-free permutations and the edges to the action of the permutations on the vertices, as in Figure 3.1.

Some of the relevant parameters of $\mathcal{G}_{cff}$ can be easily computed for existing constructions of claw-free functions. Let us consider the first scheme proposed by Goldwasser et al., defined by $f_0(s) = s^2 \mod n$, $f_1(s) = 4s^2 \bmod n$ for s a quadratic residue modulo n where $n = p_1p_2$ with primes $p_1 \equiv 3 \bmod 8$ and $p_2 \equiv 7 \bmod 8$. If the computation starts from a quadratic residue s_0 then the hash value of m is given by $H(s_0||n,m) = 4^m s_0^{2^{|m|}} \bmod n$.

The graph $\mathcal{G}_{cff}$ is directed and 2-regular. Its girth is only 1 as it has two loops in the vertices $s = 1$ and $s = 4^{-1}$, but reaching one of these two vertices from a random initial point s_0 seems a hard problem: indeed, given a message m such that $4^{-1} = 4^m s_0^{2^{|m|}}$ or $1 = 4^m s_0^{2^{|m|}}$ we get a solution to the representation problem in $\mathbb{F}_n^*$ with generators s_0 and 4. The hash value of a message of length μ belongs to the set $< 4 > \cdot s_0^{2^\mu}$ where $< 4 >$ is the subgroup of $\mathbb{F}_n^*$ generated by 4.

We point out that the distribution of random messages of increasing length μ among this set tends to uniformity *faster* than in Ramanujan graphs. Indeed, consider the values $H'(s_0||n,m) := H(s_0||n,m)s_0^{-2^\mu} = 4^m \bmod n$ on randomly chosen messages of length μ. Let q be the multiplicative order of 4 modulo n. For any μ, let k_μ and r_μ be the quotient and the rest of the division of 2^μ by q, that is $2^\mu = k_\mu q + r_\mu$.

If $m \bmod q < r_\mu$ there are $k_\mu + 1$ values in $[0, 2^\mu - 1]$ that lead to the result $H'(s_0||n, m) = 4^m \bmod n$, while if $m \bmod q \geq r_\mu$ there are k_μ values in $[0, 2^\mu - 1]$ that lead to $H'(s_0||n, m) = 4^m \bmod n$. Let π_μ be the distribution of $H'(s_0||n, m))$ in $< 4 >$ for randomly chosen messages m of length μ, and let u be the uniform distribution in $< 4 >$. By an easy computation,

$$\begin{aligned} ||\pi_\mu - u||_2^2 &= \left[\frac{k_\mu + 1}{2^\mu} - \frac{1}{q}\right]^2 r_\mu + \left[\frac{k_\mu}{2^\mu} - \frac{1}{q}\right]^2 (q - r_\mu) \\ &= \left[\frac{(k_\mu + 1)q - 2^\mu}{2^\mu q}\right]^2 r_\mu + \left[\frac{k_\mu q - 2^\mu}{2^\mu q}\right]^2 (q - r_\mu) \\ &= \frac{(q - r_\mu)^2 r_\mu + r_\mu^2 (q - r_\mu)}{2^{2\mu} q^2} = \frac{(q - r_\mu) r_\mu}{q} \frac{1}{2^{2\mu}} \\ &\leq \frac{q}{4} \frac{1}{2^{2\mu}}, \end{aligned}$$

hence the mixing rate of the graph constructed is $\rho = \lambda = \frac{1}{2}$ which is better than the optimal non-backtracking mixing rate per bit $\frac{1}{\sqrt{2}}$ obtained for undirected Ramanujan graphs. This moderates the advantage of using Ramanujan graphs in expander hashes, especially in comparison with directed graphs.

Expander hashes and claw-free permutation-based hash functions are similar at first sight, but the security arguments for these two designs are very different. In the claw-free permutation design, the function describing the edge relation cannot be inverted because of the claw-free property. In the expander hash design, this function can be inverted efficiently; the hash function becomes non invertible only after the edge function is iterated many times. In the expander hash design, much less "cryptographic strength" is set on each edge relation but on the other hand this allows for more efficient hashing schemes.

The above analysis for Goldwasser et al.'s hash function may be generalized to a class of graphs $\mathcal{G}_{DL} = (V, E)$ which we will call the DL (discrete logarithm) graphs. Let G be a cyclic group and g be a generator of this group. To each group element $g_i \in G$ we associate a vertex $v_{g_i} \in V$ and we set an edge from v_{g_1} to v_{g_2} if and only if $g_2 = g_1^2$ or $g_2 = g_1^2 g$. The analysis of the girth and expanding constant of $\mathcal{G}_{cff}$ in Goldwasser et al.'s scheme extends to the DL graphs. In particular, the mixing rate of these graphs is 1/2. DL graphs are relevant to the study of hashing schemes like the Shamir and Tauman scheme [244] that are based on modular exponentiation; they are strongly related to de Bruijn's graphs [86].

We have already pointed out that expander hashes can be seen as iterated hash functions, in particular as the Merkle-Damgård transform of a very simple compression function defined by the neighborhood relation. In a sense, the converse is also true: to any compression function f mapping $\mu + \lambda$ bits to λ bits, we can associate a 2^{μ}-regular directed graph with 2^{λ} vertices. However, for large block sizes the insight offered by this perspective is small. When the degree is very large, some relevant parameters become harder to evaluate and the girth even lacks sense: there will typically exist collisions of length 1, even if these collisions are hard to find. The expander graph design for hash functions is not really suited to regular graphs with large degrees.

This section concludes our general tour of expander hash functions, their properties and the various interpretations of these properties. In the following chapters, we will concentrate on the collision and preimage resistance of particular instances, starting with the Zémor-Tillich hash function.

Chapter 5

Cryptanalytic results on ZT hash

Since its introduction at CRYPTO'94, the Zémor-Tillich hash function (ZT hash) has kept on appealing Cryptographers by its originality, its elegance, its simplicity and its security. As any Cayley hash, uniform distribution of the outputs follows from a graphical interpretation of the hash computation, and collision resistance is strictly equivalent to an interesting group theoretical problem. The function computation can be parallelized and even the serial version is quite efficient as it only requires XOR, SHIFT and TEST operations.

There have been a few publications claiming attacks on the Zémor-Tillich hash function. However, a closer look at these papers reveals that the scheme has not been seriously threatened so far. Some of the claimed "attacks" are unpractical, creating colliding messages of unreasonable length (larger than 2^{130}). Others are trapdoor attacks that can be avoided by fixing the parameters in an appropriate way. A last, important class of attacks are subgroup attacks, damaging for particular parameters in a similar way that the RSA algorithm can be insecure if the parameters are not correctly generated. The existence of these papers may have given the function a bad reputation but it remains fundamentally unbroken today.

This chapter discusses the security of the Zémor-Tillich hash function. We first describe general results on the group $SL(2, \mathbb{F}_{2^n})$ and we give the positive security results on the function that are obtained from the graph-theoretical and group-theoretical perspectives. We then focus on preimage and collision resistance and particularly on the representation problem. We review all the cryptanalytic results published about the Zémor-Tillich hash function: we describe, analyze and in many cases improve attacks that had often only been justified by concrete examples on particular parameters or on reduced versions.

Subsequently, we identify hard and easy components of the representation problem and we deduce new collision and preimage subgroup attacks against the Zémor-Tillich hash function. Unlike previous ones, our attacks are generic in the sense that they work for any parameters of the function. With a time complexity close to $2^{n/2}$, our attacks beat by far the birthday bound and ideal preimage complexities which are $2^{3n/2}$ and 2^{3n} for the Zémor-Tillich hash function. They are practical up to $n \approx 120, 130$ that is very close to the parameter's lower bound $n \geq 130$ initially proposed by Zémor and Tillich. As the attacks include a birthday search in a reduced set of size 2^n they do not invalidate the scheme but rather suggest that the initial parameters were too small.

Our collision attacks suggest that an output of n bits should be extracted from the original $3n$ bits of Zémor-Tillich. We consequently introduce the vectorial and projective variants of Zémor-Tillich that have output sizes respectively $2n$ and n bits. We show that the original function is collision resistant if and only if its vectorial variant and (for small n) if and only if its projective variant are collision resistant. We then discuss further cryptanalytic ideas and we introduce a few problems the solution of which would break the collision resistance of ZT hash.

This chapter is the first survey of attacks against the Zémor-Tillich hash function and it also introduces significant new ideas. Parts of the results presented were obtained with Jean-Jacques Quisquater, Jean-Pierre Tillich and Gilles Zémor [207] and an early version of this chapter has benefited from a careful review and improvements by Jean-Pierre Tillich. The chapter is organized as follows. Section 5.1 presents general results that will be useful for the next sections. Section 5.2 gives positive security results for ZT hash. Section 5.3 discusses previous attacks, Section 5.4 introduces our new collision and preimage attacks and Section 5.5 describes the vectorial and projective variants of Zémor-Tillich. Section 5.6 presents further and promising approaches and Section 5.7 concludes the chapter.

5.1 On the group $SL(2, \mathbb{F}_{2^n})$ and the generators A_0 and A_1

Let $\mathbb{F}_{2^n} := \mathbb{F}_2[X]/(P_n(X))$ where $P_n(X)$ is an irreducible polynomial of degree n. We recall from Section 4.3.3 that the Zémor-Tillich hash function is the Cayley hash constructed from the group $G := SL(2, \mathbb{F}_{2^n})$ and the graph generators $S := \{A_0, A_1\}$ where

$$A_0 = \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix} \qquad A_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}.$$

In this section, we give several simple facts about the group G and the graph generators A_0 and A_1 that are useful to understand the attacks presented in the rest of this chapter together with their limitations.

5.1.1 Subgroups of $SL(2,\mathbb{F}_{2^n})$

The subgroups of $G := SL(2,\mathbb{F}_{2^n})$ are known since Dickson.

Proposition 5.1 (after [134]) *The group $G = SL(2,\mathbb{F}_{2^n})$ has order $|G| := 2^n(2^n-1)(2^n+1)$. All its proper subgroups are:*

- *for $1 \leq n' \leq n$, Abelian groups of order $2^{n'}$;*
- *cyclic groups of order dividing $2^n \pm 1$;*
- *dihedral groups of order 2 times a divisor of $2^n \pm 1$;*
- *the alternating groups A_4 and A_5 if n is even;*
- *for $1 \leq n' \leq n$, semidirect products of Abelian groups of order $2^{n'}$ with cyclic groups of order t where $t|2^{n'}-1$ and $t|2^n-1$;*
- *groups $SL(2,\mathbb{F}_{2^{n'}})$ for $n'|n$*
- *groups $PGL(2,\mathbb{F}_{2^{n'}})$ for $2n'|n$.*

We see that the number and the sizes of the subgroup of $SL(2,\mathbb{F}_{2^n})$ highly depend on the parameter n. Subgroups and elements of small order are particularly interesting for Zémor-Tillich cryptanalysts. The group G always contains elements of order 2 and 3, a property used in the trapdoor attack of Section 5.3.3. Elements of order 2 or 3 can be recognized from their traces.

Proposition 5.2 (after [251]) *A matrix $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in SL(2,\mathbb{F}_{2^n})$ has order 2 if and only if $t := a+d = 0$. It has order 3 if and only if $t = 1$.*

Proof: For any matrix $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, we have $P_M(M) = 0$ where $P_M(\lambda) = \lambda^2 + (a+d)\lambda + (ad+bc) = \lambda^2 + t\lambda + 1$ is the characteristic polynomial of M [160]. Hence we have

$$\begin{aligned} \mathrm{ord}(M) = 2 &\Leftrightarrow I \neq M \text{ and } I = M^2 = tM + I \\ &\Leftrightarrow I \neq M \text{ and } t = 0, \\ \mathrm{ord}(M) = 3 &\Leftrightarrow I \neq M \text{ and } I = M^3 = tM^2 + M = (t^2+1)M + I \\ &\Leftrightarrow I \neq M \text{ and } t = 1. \end{aligned}$$

□

When n is composite, the trace of a matrix $M \in G$ also determines whether M belongs to the subgroup $SL(2, \mathbb{F}_{2^{n'}})$ or a conjugate subgroup. Let us write n' for a factor of n.

Proposition 5.3 [251]

- *Let $n'|n$ and $M \in SL(2, \mathbb{F}_{2^n})$. Then M is similar to a matrix $M' \in SL(2, \mathbb{F}_{2^{n'}})$ if and only if $Trace(M) \in \mathbb{F}_{2^{n'}}$.*
- *For $M \in SL(2, \mathbb{F}_{2^n})$ with $Trace(M) \in \mathbb{F}_{2^{n'}}$ we have $ord(M) \leq 2^{n'} + 1$.*

Some subgroups of $SL(2, \mathbb{F}_{2^n})$ exist independently of the value of n: the diagonal subgroup, the triangular subgroups, or the subgroups of matrices with a given eigenvector. Proposition 5.4 describes these subgroups and decomposes the group operation inside these subgroups into an Abelian and a non-Abelian parts.

Proposition 5.4

1. *Let $\mathcal{D}$ be the subgroup of unimodular diagonal matrices with elements in $\mathbb{F}^*_{2^n}$. Then $\mathcal{D} = \{D_a := \left(\begin{smallmatrix} a & \\ & a^{-1} \end{smallmatrix}\right), \forall a \in \mathbb{F}^*_{2^n}\}$. Moreover, for all $D_{a_1}, D_{a_2} \in \mathcal{D}$, we have $D_{a_1}D_{a_2} = D_{a_2}D_{a_1} = D_{a_1a_2}$.*
2. *Let $\mathcal{T}^{up}$ be the subgroup of unimodular upper triangular matrices with elements in $\mathbb{F}_{2^n}$. Then $\mathcal{T}^{up} = \{T_{a,b} := \left(\begin{smallmatrix} a & b \\ & a^{-1} \end{smallmatrix}\right), \forall a \in \mathbb{F}^*_{2^n}, b \in \mathbb{F}_{2^n}\}$. Moreover, for all $T_{a_1,b_1}, T_{a_2,b_2} \in \mathcal{T}^{up}$ we have $T_{a_1,b_1}T_{a_2,b_2} = T_{(a_1a_2),(a_1b_2+b_1a_2^{-1})}$ and for all $b \in \mathbb{F}_{2^n}$ we have $T^2_{1,b} = I$.*
3. *For any vector $v = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right) \in \mathbb{F}^{1\times 2}_{2^n}$, let $\mathcal{L}^v$ be the subgroup of unimodular matrices with elements in $\mathbb{F}_{2^n}$ that have v as a left eigenvector. Then $\mathcal{L}^v = \{M_{\lambda,\alpha,\beta} := \lambda I + \left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right)\left(\begin{smallmatrix} \alpha & \beta \end{smallmatrix}\right), \forall \lambda \in \mathbb{F}^*_{2^n}, \alpha, \beta \in \mathbb{F}_{2^n}$ s.t. $\det(M_{\lambda,\alpha,\beta}) = \lambda^2 + \lambda(\alpha b + a\beta) = 1\}$. In particular if $M_{1,\alpha,\beta} \in$*

$\mathcal{L}^v$ then $\left(\begin{smallmatrix} \alpha & \beta \end{smallmatrix} \right) = \delta \left(\begin{smallmatrix} a & b \end{smallmatrix} \right)$ for some $\delta \in \mathbb{F}_{2^n}$.Moreover, for all $M_{\lambda_1,\alpha_1,\beta_1}, M_{\lambda_2,\alpha_2,\beta_2} \in \mathcal{V}^l_v$, we have $M_{\lambda_1,\alpha_1,\beta_1} M_{\lambda_2,\alpha_2,\beta_2} = M_{(\lambda_1\lambda_2),(\alpha_1+\alpha_2+\alpha_1\alpha_2 b)+a\alpha_2\beta_1,(\beta_1+\beta_2+\beta_1\beta_2 a)+b\alpha_1\beta_2}$ and for all $\delta \in \mathbb{F}_{2^n}$ we have $M^2_{1,\delta a,\delta b} = I$.

Proof: Most of the proposition is easy; we give the proof that $M \in \mathcal{L}^v$ can be written as $\lambda I + \left(\begin{smallmatrix} b \\ a \end{smallmatrix} \right) \left(\begin{smallmatrix} \alpha & \beta \end{smallmatrix} \right)$ for some $\lambda \in \mathbb{F}^*_{2^n}, \alpha, \beta \in \mathbb{F}_{2^n}$. Let $v = \left(\begin{smallmatrix} a & b \end{smallmatrix} \right) \in \mathbb{F}^{1\times 2}_{2^n}$ and $M \in \mathcal{L}^v$. Let $c, d \in \mathbb{F}_{2^n}$ such that $ad + bc = 1$. We can write $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix} \right) M = \left(\begin{smallmatrix} a & b \\ c' & d' \end{smallmatrix} \right) \lambda$ for some $\lambda \in \mathbb{F}^*_{2^n}$ and $c', d' \in \mathbb{F}_{2^n}$ such that $ad' + bc' = \lambda^{-1}$. Then $M = \lambda \left(\begin{smallmatrix} d & b \\ c & a \end{smallmatrix} \right) \left(\begin{smallmatrix} a & b \\ c' & d' \end{smallmatrix} \right) = \lambda \left(\begin{smallmatrix} 1+b(c+c') & b(d+d') \\ a(c+c') & 1+a(d+d') \end{smallmatrix} \right) = \lambda I + \left(\begin{smallmatrix} b \\ a \end{smallmatrix} \right) \left(\begin{smallmatrix} \alpha & \beta \end{smallmatrix} \right)$ where $\alpha := \lambda(c + c')$ and $\beta := \lambda(d + d')$. □

Similar results can be derived for the subgroup of lower triangular matrices and the subgroups of matrices with a given right eigenvector. Proposition 5.4 will be used to prove Proposition 5.10, which in turn inspires our attacks of Sections 5.4.

5.1.2 Homomorphism from $SL(2, \mathbb{F}_2[X])$

Let $\widetilde{A}_0$ and $\widetilde{A}_1$ be the matrices A_0 and A_1 viewed as elements of $SL(2, \mathbb{F}_2[X])$ rather than as elements of $SL(2, \mathbb{F}_{2^n})$. The matrices $\widetilde{A}_0$ and $\widetilde{A}_1$ generate a subset Ω of $SL(2, \mathbb{F}_2[X])$

$$\Omega = \langle \widetilde{A}_0, \widetilde{A}_1 \rangle := \left\{ \widetilde{M} = \prod_i \widetilde{M}_i \text{ s.t. } \widetilde{M}_i \in \left\{ \widetilde{A}_0, \widetilde{A}_1 \right\} \right\}$$

and there is a natural homomorphism $\varphi : \Omega \to SL(2, \mathbb{F}_{2^n})$ defined by the "reduction modulo $P_n(X)$".

Proposition 5.5 *[251] Each element $\widetilde{M} \in \Omega$ has a unique factorization as a product of $\widetilde{A}_0$ and $\widetilde{A}_1$. Moreover, $\widetilde{M}$ has the form*

$$\begin{pmatrix} a_\mu(X) & b_{\mu-1}(X) \\ c_{\mu-1}(X) & d_{\mu-2}(X) \end{pmatrix} \text{ or } \begin{pmatrix} a_\mu(X) & b_\mu(X) \\ c_{\mu-1}(X) & d_{\mu-1}(X) \end{pmatrix}$$

depending on whether the last right factor is $\widetilde{A}_0$ or $\widetilde{A}_1$, where μ is the number of factors (the subscript indices are the degrees of the polynomials).

Proof: For $\mu = 1$ the proposition is trivial. Assuming it is true for some value μ, proving it is also true for $\mu + 1$ amounts to computing the four products

$$\begin{pmatrix} a_\mu & b_{\mu-1} \\ c_{\mu-1} & d_{\mu-2} \end{pmatrix} \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} a_\mu & b_{\mu-1} \\ c_{\mu-1} & d_{\mu-2} \end{pmatrix} \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}$$
$$\begin{pmatrix} a_\mu & b_\mu \\ c_{\mu-1} & d_{\mu-1} \end{pmatrix} \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} a_\mu & b_\mu \\ c_{\mu-1} & d_{\mu-1} \end{pmatrix} \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}$$

and checking that the entries of the resulting matrices have the right degrees. □

Factoring a matrix $\widetilde{M} \in \Omega$ is easy: the right last factor $\widetilde{A}_{m_\mu}$ of $\widetilde{M}$ is given by Proposition 5.5, then the second right last factor of $\widetilde{M}$ is the right last factor of $\widetilde{M}\widetilde{A}_{m_\mu}^{-1}$, and so on.

The homomorphism φ cannot be inverted today, as we cannot characterize the set Ω by a few equations, nor characterize any set Ω' with $\Omega \subset \Omega' \subset SL(2, \mathbb{F}_2[X])$ and $|\Omega'|/|\Omega|$ small. For example, if we define

$$E_\mu := \left\{ M = \left(\begin{smallmatrix} \widetilde{a}(X) & \widetilde{b}(X) \\ \widetilde{c}(X) & \widetilde{d}(X) \end{smallmatrix} \right) \in SL(2, \mathbb{F}_2[X]) \text{ s.t. } \deg \widetilde{a}, \widetilde{b}, \widetilde{c}, \widetilde{d} \leq \mu \right\}$$

then [258]

$$\frac{|\Omega \cap E_\mu|}{|E_\mu|} = O(2^{-\mu}).$$

As we will see in Section 5.6.1 and Chapter 6, the situation differs greatly in Zémor's first proposal and in LPS and Morgenstern hash functions, allowing lifting attacks [260, 259, 205] that could not be extended to the Zémor-Tillich hash function.

5.1.3 On powers of elements

Although the set Ω is badly understood, one of its subset such as $\left\{ \widetilde{A}_0^\mu, \mu \in \mathbb{Z}^+ \right\}$ has been well characterized, as can be the set $\left\{ \widetilde{A}^\mu, \mu \in \mathbb{Z}^+ \right\}$ for any matrix $A \in SL(2, \mathbb{F}_2[X])$.

Define the polynomials $f_i(X)$ as $f_{-1}(X) = 1, f_0(X) = 0, f_1(X) = 1$ and $f_{i+2}(X) = X f_{i+1}(X) + f_i(X)$ for $i \geq 1$. By induction [18],

$$\widetilde{A}_0^\mu(X) = \left(\begin{smallmatrix} f_{\mu+1}(X) & f_\mu(X) \\ f_\mu(X) & f_{\mu-1}(X) \end{smallmatrix} \right). \tag{5.1}$$

Similarly, as $\widetilde{A}_0^{-1}(X)\widetilde{A}_1(X)\widetilde{A}_0(X) = \left(\begin{smallmatrix} X+1 & 1 \\ 1 & 0 \end{smallmatrix} \right) = \widetilde{A}_0(X+1)$, we have

$$\widetilde{A}_1(X)^\mu = \widetilde{A}_0(X) \left(\begin{smallmatrix} f_{\mu+1}(X+1) & f_\mu(X+1) \\ f_\mu(X+1) & f_{\mu-1}(X+1) \end{smallmatrix} \right) \widetilde{A}_0^{-1}(X). \tag{5.2}$$

The polynomials $f_i(X)$ have interesting properties pointed out by [18]. Let $\lambda_0, \lambda_0^{-1}$ be the eigenvalues of $\widetilde{A}_0$. Then $f_\mu(X) = \frac{1}{X} Tr\left(\widetilde{A}_0^\mu(X)\right) = \frac{1}{X}\left(\lambda_0^\mu + \lambda_0^{-\mu}\right)$ and in particular $f_{2^\mu}(X) = X^{2^\mu - 1}$. If $\mu > 0$ is even then

$f_\mu(X) = Xg_\mu(X)^2$ for some polynomial $g_\mu(X) \in \mathbb{F}_2[X]$ and if $\mu > 0$ is odd then $f_\mu(X) = h_\mu(X)^2$ for some polynomial $h_\mu(X) \in \mathbb{F}_2[X]$.

The following proposition extends Equations (5.1) and (5.2).

Proposition 5.6 *Let $\widetilde{A} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in SL(2, \mathbb{F}_2[X])$ and let $t := a + d$. Then $\widetilde{A}^\mu = f_{\mu-1}(t)I + f_\mu(t)\widetilde{A}$. In particular, if $\widetilde{A} = \left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right)$ then $\widetilde{A}^k = \left(\begin{smallmatrix} f_{\mu+1}(t) & f_\mu(t) \\ f_\mu(t) & f_{\mu-1}(t) \end{smallmatrix}\right)$.*

PROOF: For $\mu = 0, 1$ the result is trivial. It is true for $\mu = 2$ as $\widetilde{A}^2 = \left(\begin{smallmatrix} a^2+bc & ab+bd \\ ac+cd & bc+d^2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) + t \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ because $ad + bc = 1$. Now suppose it is true for any integer $\leq \mu$. Then $\widetilde{A}^{k+1} = \widetilde{A}(f_{\mu-1}(t)I + f_\mu(t)\widetilde{A}) = f_{\mu-1}(t)\widetilde{A} + f_\mu(t)\widetilde{A}^2 = f_\mu(t)I + (f_{\mu-1}(t) + tf_\mu(t))\widetilde{A} = f_\mu(t)I + f_{\mu+1}(t)\widetilde{A}$ so we are done with the first part of the proposition. The second part follows immediately. □

A different characterization can be obtained from the Jordan decompositions of the matrices. Let λ, λ' be the eigenvalues of $\widetilde{A} \in \Omega \subset SL(2, \mathbb{F}_2[X])$. Note that $\lambda' = \lambda^{-1} \neq \lambda$ as $\lambda\lambda' = \det(\widetilde{A}) = 1$ and $\lambda + \lambda' = Tr(\widetilde{A}) = a + d$ where $\deg(a) \neq \deg(d)$ according to Proposition 5.5. Consequently, $\widetilde{A}$ decomposes as $\widetilde{A} = S \left(\begin{smallmatrix} \lambda & 0 \\ 0 & \lambda^{-1} \end{smallmatrix}\right) S^{-1}$ with $S = \left(\begin{smallmatrix} b & b \\ a+\lambda & a+\lambda^{-1} \end{smallmatrix}\right)$ and $S^{-1} = b^{-1}(a+d)^{-1} \left(\begin{smallmatrix} a+\lambda^{-1} & b \\ a+\lambda & b \end{smallmatrix}\right)$. Finally, the powers of $\widetilde{A}$ can be written as

$$\widetilde{A}^\mu = S \left(\begin{smallmatrix} \lambda^\mu & 0 \\ 0 & \lambda^{-\mu} \end{smallmatrix}\right) S^{-1}. \tag{5.3}$$

We conclude this section with an observation of Geiselmann that the powers of A_0 and A_1 can be embedded into finite fields [111].

Proposition 5.7 [111] *Let $A = \left(\begin{smallmatrix} t & 1 \\ 1 & 0 \end{smallmatrix}\right) \in SL(2, \mathbb{F}_{2^n})$ and its characteristic polynomial $P_A(\lambda) := \lambda^2 + t\lambda + 1 \in \mathbb{F}_{2^n}[\lambda]$.*

- *If $P_A(\lambda)$ is irreducible, then we have isomorphisms $\mathbb{F}_{(2^n)^2} \simeq \mathbb{F}_{2^n}/(P_A(\lambda)) \simeq \mathbb{F}_{2^n}.I + \mathbb{F}_{2^n}.A$. Moreover, a matrix $M \in \mathbb{F}_{2^n}^{2\times 2}$ is a power of A if and only if $M = \alpha I + \beta A$ for some $\alpha, \beta \in \mathbb{F}_{2^n}$, $\det(M) = 1$ and $M^{ord(A)} = I$.*

- *If $P_A(\lambda)$ factorizes into $P_A(\lambda) = (\lambda + \lambda_0)(\lambda + \lambda_0^{-1})$, then the relation $\sim$ defined by $\alpha_1 I + \beta_1 A \sim \alpha_2 I + \beta_2 A \Leftrightarrow \alpha_1 + \beta_1\lambda_0 = \alpha_2 + \beta_2\lambda_0$ is an equivalence relation.*
  The application $\varphi : \overline{\alpha I + \beta A} \rightarrow \alpha + \beta\lambda_0$ where $\overline{\alpha I + \beta A}$ is the equivalence class of $\alpha I + \beta A$, is a homomorphism. A matrix $M \in \mathbb{F}_{2^n}^{2\times 2}$ is a power of A if and only if $M = \alpha I + \beta A$ for some $\alpha, \beta \in \mathbb{F}_{2^n}$, $\det(M) = 1$ and $(\alpha + \beta\lambda_0)^{ord(A)} = 1$.

Proposition 5.7 is used in Geiselmann's attack (Section 5.3.4) to characterize the powers of $A_0 = \left(\begin{smallmatrix} X & 1 \\ 1 & 0 \end{smallmatrix}\right)$ but also $A_1 = \left(\begin{smallmatrix} X & X+1 \\ 1 & 1 \end{smallmatrix}\right)$. Indeed, the proposition applies to $A_0' := \left(\begin{smallmatrix} X+1 & 1 \\ 1 & 0 \end{smallmatrix}\right) = A_0^{-1}A_1A_0$ and hence it extends to A_1 because

- $(A_0')^\mu = \alpha I + \beta A_0' \Leftrightarrow A_1^\mu = A_0(\alpha I + \beta A_0')A_0^{-1} = \alpha I + \beta A_1$,
- $\det(\alpha I + \beta A_0') = \det(\alpha I + \beta A_1)$
- $(\alpha I + \beta A_0')^{ord(A_0')} = (\alpha I + \beta A_1)^{ord(A_1)}$ and $(\alpha + \beta\lambda_0)^{ord(A_0')} = (\lambda_0(\alpha + \beta\lambda_0)\lambda_0^{-1})^{ord(A_1)}$.

5.2 Positive security results on ZT hash

In this section, we give the security properties of ZT hash that can be derived from the graph-theoretical and group-theoretical perspectives developed in Chapter 5. For $G := SL(2, \mathbb{F}_{2^n})$ and $S := \{A_0, A_1\}$, the Cayley graph $\mathcal{ZT} := \mathcal{C}_{G,S}$ will be called the Zémor-Tillich graph. An example of such a graph is represented in Figure 5.1.

$\mathcal{ZT}$ is a strongly connected graph: the set of matrices generated by A_0 and A_1 is the whole group G. The proof in [258] considers all possible subgroups of G (see Proposition 5.1) and shows that none of these subgroups contains both A_0 and A_1.

The girth of $\mathcal{ZT}$ is at least n [258]. Indeed, let $\widetilde{A}_0$ and $\widetilde{A}_1$ be the matrices A_0 and A_1 viewed as elements of $SL(2, \mathbb{F}_2[X])$ rather than as elements of $SL(2, \mathbb{F}_{2^n})$. Proposition 5.5 implies that the set $\Omega :=< \widetilde{A}_0, \widetilde{A}_1 >$ is free, hence if $m_0...m_{\mu-1} \neq m_0'...m_{\mu'-1}'$ the corresponding products $\widetilde{M} := \widetilde{A}_{m_0}...\widetilde{A}_{m_{\mu-1}}$ and $\widetilde{M'} := \widetilde{A}_{m_0'}...\widetilde{A}_{m_{\mu'-1}'}$ cannot be equal in $SL(2, \mathbb{F}_2[X])$. Consequently, if two products $M := A_{m_0}...A_{m_{\mu-1}}$ and $M' := A_{m_0'}...A_{m_{\mu'-1}'}$ are equal in $SL(2, \mathbb{F}_{2^n})$, there must be some reduction modulo $P_n(X)$ happening, hence at least one of $\widetilde{M}$ and $\widetilde{M'}$ has at least one of its entries of degree at least n. By Proposition 5.5, this implies that $\max(\mu, \mu') \geq n$ which by definition shows that the girth is at least n.

The diameter of $\mathcal{ZT}$ is expected to be small although this has not been proved. If Babai's conjecture (Conjecture 4.1) is true even for non symmetric generating sets, the diameter of $\mathcal{ZT}$ is a polynomial function of the degree n. Of course, a constructive proof of this conjecture would certainly break any Cayley hash constructed from non-Abelian simple linear groups but at the present even non-constructive proofs seem out of reach in the general case [129]. In particular, Zémor's proof for his first function [274, 275] and

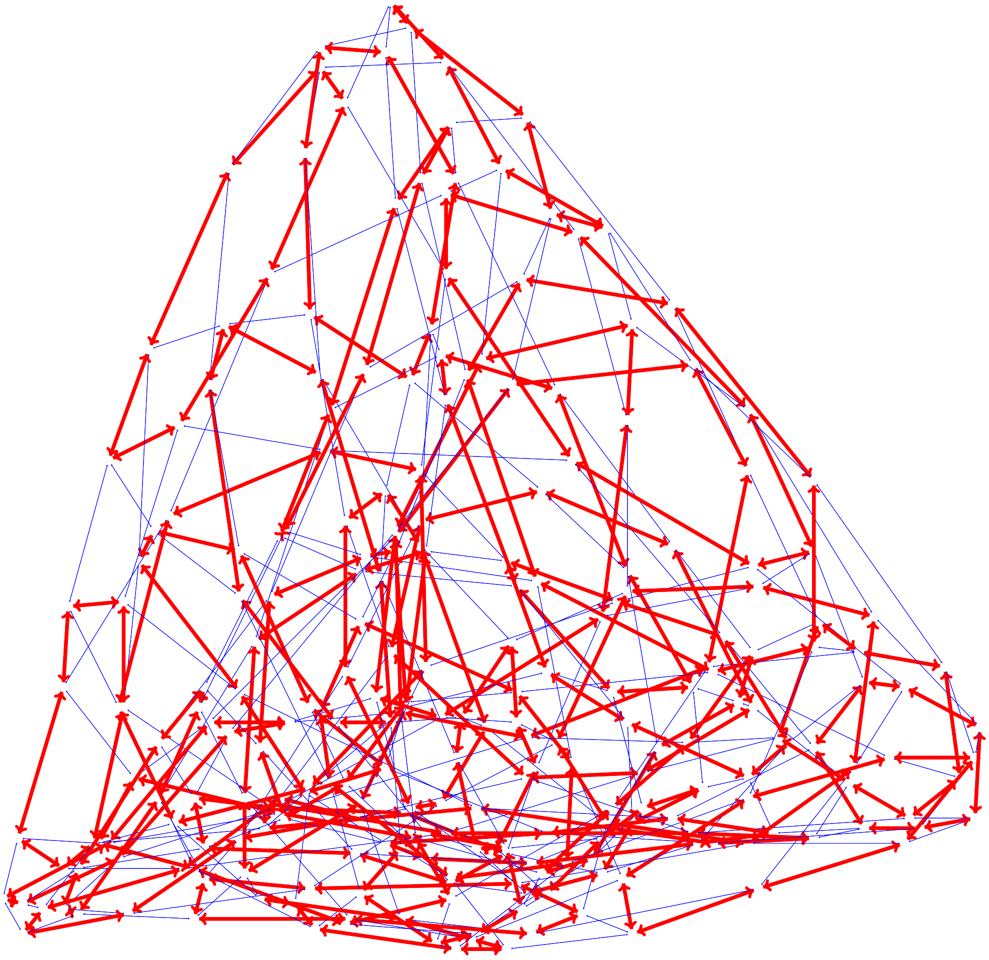

Figure 5.1: $\mathcal{ZT}$ graph for parameter $P_2(X) = X^3+X^2+1$. (Due to a symmetry in the eigenvectors of the graph, many couple of vertices are superposed in this graph representation, resulting that many edges look undirected.)

Helfgott's proof for undirected Cayley graphs of $SL(2,\mathbb{F}_p)$ [129] are both non-constructive.

No good bound is known on the expanding constant nor the second eigenvalue of $\mathcal{ZT}$ but the distribution of hash values tends to equidistribution when the message length tends to infinity. Using Proposition 5.4, it is possible to prove that the greatest common multiple of all the cycle lengths of $\mathcal{ZT}$ is 1 [258], which by Theorem 4.2 implies that $\lambda(\mathcal{ZT}) < 1$.

5.3 Previous results on ZT hash

We now review previous cryptanalytic results on the Zémor-Tillich hash function. The function can be inverted for short messages (Section 5.3.1). Public or secret collisions can be forced by the adversary if he can choose the parameter $P_n(X)$ (Sections 5.3.2 and 5.3.3). Collisions, although very large, can be computed by solving discrete logarithm problems (Section 5.3.4). Finally, the subgroup structure can be exploited to produce collisions with more or less success depending on the parameters (Sections 5.3.5 and 5.3.6). The "attacks" presented here are known [70, 111, 18, 251] but we have cleaned their exposition, analyzed their efficiency, discussed their practical impact and in many cases extended them.

5.3.1 Invertibility for short messages

The invertibility for short messages was first observed in [251]. If the message size μ is smaller than n, then according to Proposition 5.5 all the entries of the hash are polynomials with degree smaller than n. Consequently, no polynomial reduction is applied and the hash output can be seen as an element of $\Omega \subset SL(2, \mathbb{F}_2[X])$ (see Section 5.1.2). As factorization is easy in this set, the hash function can be inverted. Invertibility on short messages does not contradict preimage resistance for large message sets but it discards the function for certain applications.

In particular, suppose that a Zémor-Tillich hash function of size $n = 170$ is used to hash an ECDLP key K of $\mu = 160$ bits, and the resulting value $H_{ZT}(K)$ is intercepted by some adversary: then the adversary can recover K as above. This attack can be extended to larger keys of size μ slightly larger than n: indeed, the adversary may guess the $\mu - n + 1$ last bits of the key and recover the remaining bits as before (discarding the keys when the matrices do not have the correct form). It is reasonable to expect this attack to be feasible today if $\mu < n + 60$, and prudent to take as security margin $\mu > n + 120$.

5.3.2 Charnes-Pieprzyck attack

Charnes and Pieprzyck [70] were the first to identify an attack against the Zémor-Tillich hash function. This attack was better analyzed by Abdukhalikov et Kim [18]; we follow mainly their exposition.

Let $f_i(X)$ be the polynomials defined in Section 5.1.3. The attacker may choose $P_n(X)$ such that $P_n(X)|f_\mu(X)$ for some small μ. According to

Equation (5.1),

$$\widetilde{A_0}(X)^{\mu} = \left(\begin{smallmatrix} f_{\mu+1}(X) & f_{\mu}(X) \\ f_{\mu}(X) & f_{\mu-1}(X) \end{smallmatrix} \right)$$

and as $\det\left(\widetilde{A_0}^{\mu}\right) = \det\left(\widetilde{A_0}\right)^{\mu} = f_{\mu+1}(X)f_{\mu-1}(X) + f_{\mu}(X)^2 = 1$, the property $P_n(X)|f_{\mu}(X)$ implies $f_{\mu+1}(X) = f_{\mu-1}(X) = 1 \bmod P_n(X)$. Finally, we obtain the equality

$$A_0^{\mu} = I \bmod P_n(X)$$

which gives a collision of size μ. Similarly from Equation (5.2), if the attacker chooses $P_n(X)$ such that it divides $f_{\mu'}(X+1)$ for some small μ' then there is a collision of size μ' given by $A_1^{\mu'} = I \bmod P_n(X)$.

Charnes-Pieprzyck's attack can be extended as follows. From any short message m of size μ_1, let $\widetilde{A}(X) = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix} \right)$ be its hash value viewed as a matrix in $SL(2, \mathbb{F}_2[X])$, and let $P_n(X)$ be such that $P_n(X)|f_{\mu_2}(a+d)$ for some small μ_2. As $\widetilde{A}$ and $\widetilde{A}' := \left(\begin{smallmatrix} a+d & 1 \\ 1 & 0 \end{smallmatrix} \right)$ have the same characteristic polynomial, we can write $\widetilde{A} = S\widetilde{A}'S^{-1}$ for some S. Applying Proposition 5.6,

$$\widetilde{A}^{\mu_2} = (S\widetilde{A}'S^{-1})^{\mu_2} = S(\widetilde{A}')^{\mu_2}S^{-1} = SS^{-1} = I \bmod P_n(X),$$

that is the message m repeated μ_2 times produces a message of size $\mu_1\mu_2$ colliding with the void message.

In Charnes-Pieprzyck's attack, a collision of the form $(m^{\mu}, void)$ for m equal to 0 or 1 is produced, while in our extension m can be any message. In Charnes-Pieprzyck's attack, the collision could hardly be kept secret as anybody can check the orders of A_0 and A_1. In our extension, the collision remains secret and hence may be used as a trapdoor, that is as a secret information useful to perform some computation otherwise infeasible, in this case to produce collisions (see Section 3.1.1).

These attacks point out undesired properties of the Zémor-Tillich hash function but their practical impact is weak as they can easily be avoided. If the polynomial is fixed randomly, let us say by some authority, the probability that $P_n(X)$ divides $f_{\mu}(X)$ or $f_{\mu}(X+1)$ for small μ is clearly very small (bounds are given by Abdukhalikov et Kim [18]) so Charnes-Pieprzyck's attack will be defeated. Our extension will also be defeated: the condition $P_n(X)$ divides $f_{\mu}(t(X))$ for some $t(X)$ may seem easier to satisfy if $t(X)$ is not fixed, but then finding a message hashing to a matrix in Ω with a given trace $t(X)$ seems to be a hard problem.

5.3.3 Steinwandt et al.'s trapdoor attack

In the trapdoor attack of [251], the polynomial $P_n(X)$ is chosen from a message m such that $m||m$ or $m||m||m$ collides with the void message. For a given message $m = m_0...m_{\mu-1}$ let

$$\widetilde{A} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) := \widetilde{A}_{m_0}\widetilde{A}_{m_1}...\widetilde{A}_{m_{\mu-1}},$$

i.e., where the hash value is computed in the ring $\mathbb{F}_2[X]$ rather than the field $\mathbb{F}_{2^n}$. If $P_n(X)$ is one of the irreducible factors of $a+d$ (resp. $a+d+1$) then according to Proposition 5.2 the message $m||m$ (resp. $m||m||m$) collides with the void message. As an example, [251] gives a polynomial of degree 167 such that the message "*This is the way a trapdoor can look like.* " repeated twice collides with the the void message.

When n has small factors the attack can be extended to $H(m)$ of small order different of 2 or 3. Indeed, Proposition 5.3 implies that any matrix $A = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in SL(2, \mathbb{F}_{2^n})$ with trace $t \in \mathbb{F}_{2^{n'}} \subset \mathbb{F}_{2^n}$ has order smaller than $2^{n'}+1$. Consequently, if $P_n(X)$ is a factor of $a+d+t$ then the message m repeated $\mathrm{ord}(A) \leq 2^{n'}+1$ times collides with the void message.

For a given message m, the basic attack succeeds if and only if there exists an irreducible factor of $a+d$ or of $a+d+t$ with degree $130 \leq n \leq 170$. The number of irreducible polynomials of degree n over $\mathbb{F}_2$ is given by $N(n) = \frac{1}{n}\sum_{d|n}\mu\left(\frac{n}{d}\right)2^d$ where μ is the Moebius function; from numerical calculations the first terms of this sequence may be approximated by $N(n) \approx 2^{n-6}$ [269]. The probability that a given polynomial of degree n divides a random polynomial of large degree is $1/2^n$; the probability that there exists one polynomial of degree n dividing a random polynomial of large degree is $1-\left(1-\frac{1}{2^n}\right)^{2^{n-6}} \approx 1-e^{-2^{-6}} \approx 0.0145$. Finally, the probability that a random polynomial of large degree has an irreducible factor with degree $130 \leq n \leq 170$ is about $1-(1-0.0145)^{40} \approx 0.4647$.

The attack will fail for both $a+d$ and $a+d+1$ only for about 29% of messages. In applications where ASCII coded English messages are hashed, the adversary may add blank characters to the message or replace a word with a synonym until he finds a trapdoor. If n is composite, more polynomials $a+d+t$ may be considered so the probability of success increases and less trials are needed.

As the attack produces collisions on most arbitrary messages, its effects may be devastating. Consider the case of a contract concluded with an electronic protocol using the hash-and-sign paradigm (Section 2.6.2) and the Zémor-Tillich hash function. Suppose Bob had the opportunity to choose $P_n(X)$ such that the message *"I also agree to pay Alice 10,000 dollars the following day.* " collides with the void message. Bob can then sign the message *I Bob agree on paying 10,000 dollars to Alice on date 07/23/2009. I also agree to pay Alice 10,000 dollars on the following day. I also agree to pay Alice 10,000 dollars on the following day.* " and later claim that he had actually only signed the first sentence: the signature scheme would not be undeniable.

Despite of this scenario and many similar ones, we argue that the trapdoor attack will have little or no practical impact if the polynomial is fixed once and for all by some authority. Indeed, even if the authority had chosen a parameter with a trapdoor, it could use it only to create collisions on certain pre-fixed messages. In particular, the trapdoor would be of no help for inverting the hash function nor for finding second preimages. Moreover, any use of the collision would reveal it, so not only the attack could be performed only once but the authority would immediately look very suspicious, as it would be very unlikely that this collision was generated honestly. Finally, we point out that some choices of parameters discard trapdoor attacks even by the authority, for example if $P_n(X)$ is the smallest polynomial of some degree n or if it depends on the binary representation of a universal constant like π.

5.3.4 Geiselmann's "attack"

Geiselmann proposed an approach for constructing collisions valid for any choice of the parameter n [111]. The "attack" requires solving discrete logarithms in the field $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$, which is possible for the values $130 \leq n \leq 170$ proposed by Zémor and Tillich. Its main drawback is to produce very long collisions (actually larger than the size of trivial collisions $A_0^{ord(A_0)} = I$) of the very special form $A_0^{e1} A_1^{e2} A_0^{e3} A_1^{e4} = M$ for some matrix M with a known short factorization.

The main idea is to use the isomorphisms φ_{A_i} of Proposition 5.7 that embeds matrix powers into a finite field K where K is either $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$. The attack has two main steps: the attacker first finds matrices A_i^{ej} satisfying $A_0^{e1} A_1^{e2} A_0^{e3} A_1^{e4} = M$, then he recovers the exponents as the discrete logarithms of $\varphi_{A_i}(A_i^{e_j})$ in the bases $\varphi_{A_i}(A_i)$.

The attack goes as follows. A matrix M is generated as a random short product of A_0 and A_1. A matrix equation of the form $(\alpha_1 I + \beta_1 A_0)(\alpha_2 I + \beta_2 A_1)(\alpha_3 I + \beta_3 A_0)(\alpha_4 I + \beta_4 A_1) = M$ is considered which gives 4 polynomial equations with 8 unknowns $\alpha_i, \beta_i \in \mathbb{F}_{2^n}, i = 1...4$. After adding the 4 equations $\det(\alpha_j I + \beta_j A_i) = 1$ the system is solved. In general it has solutions; otherwise another matrix M is generated. The conditions with the orders in Proposition 5.7 are checked on $\alpha_j I + \beta_j A_i$. If they are not fulfilled, another matrix M is selected, otherwise according to Proposition 5.7, $\alpha_j I + \beta_j A_i = A_i^{ej}$ for some e_j. The exponents e_j are recovered as the discrete logarithms of the matrices $\alpha_1 I + \beta_1 A_0$, $\alpha_2 I + \beta_2 A_1$, *etc.*, regarded as elements of $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$. As Zémor-Tillich prescribed parameters were $130 \leq n \leq 170$, these discrete logarithms can be computed efficiently.

We now modify Geiselmann's approach to make it more explicit. Let $\lambda_0 \in K_0$ be a root of $\lambda^2+X\lambda+1=0$, the other root being $\lambda_0' = \lambda_0^{-1} = X+\lambda_0$, and let $\lambda_1 \in K_1$ be a root of $\lambda^2+(X+1)\lambda+1=0$, the other root being $\lambda_1' = \lambda_1^{-1} = X+1+\lambda_1$. The fields K_0 and K_1 are $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$ depending on the irreducibility of the corresponding polynomials in $\mathbb{F}_{2^n}$. We can write $A_0 = S_0D_0S_0^{-1}$ where $D_0 = \begin{pmatrix} \lambda_0 & \\ & \lambda_0^{-1} \end{pmatrix}$, $S_0 = \begin{pmatrix} 1 & 1 \\ \lambda_0' & \lambda_0 \end{pmatrix}$, $S_0^{-1} = X^{-1}\begin{pmatrix} \lambda_0 & 1 \\ \lambda_0' & 1 \end{pmatrix}$ and $A_1 = S_1D_1S_1^{-1}$ where $D_1 = \begin{pmatrix} \lambda_1 & \\ & \lambda_1^{-1} \end{pmatrix}$, $S_1 = \begin{pmatrix} 1+\lambda_1 & 1+\lambda_1' \\ 1 & 1 \end{pmatrix}$, $S_1^{-1} = (X+1)^{-1}\begin{pmatrix} 1 & 1+\lambda_1' \\ 1 & 1+\lambda_1 \end{pmatrix}$.

Now $A_0^{e_1}A_1^{e_2}A_0^{e_3}A_1^{e_4} = S_0D_0^{e_1}S_0^{-1}S_1D_1^{e_2}S_1^{-1}S_0D_0^{e_3}S_0^{-1}S_1D_1^{e_4}S_1^{-1}$ so the matrix equation $A_0^{e_1}A_1^{e_2}A_0^{e_3}A_1^{e_4} = M$ corresponds to a system of four equations in the variables $x_1 := \lambda_0^{e_1}$, $x_2 := \lambda_1^{e_2}$, $x_3 := \lambda_0^{e_3}$ and $x_4 := \lambda_1^{e_4}$. If the system has no solution, larger matrix products may be considered to increase the number of variables while keeping the number of equations constant. After a solution for the variables x_i is found, the exponents e_i are recovered by computing discrete logarithms in the fields K_0 and K_1.

In both Geiselmann's abstract approach and our concrete variant, alternative products like $A_0^{e_1}A_1A_0^{e_2}A_1A_0^{e_3}A_1A_0^{e_4}A_1$ may also be considered, potentially giving systems that are easier to solve.

Neither Geiselmann's approach nor our variant can really be considered as practical attacks as the expected size of the collisions produced is very large, even larger than the expected size of trivial collisions like $A_0^{\mathrm{ord}(A_0)} = I$. Indeed, nothing prevents the exponents e_i from being of full size that is the size of the fields K_i.

5.3.5 Steinwandt et al. subgroup attacks

Depending on the parameters, the group $G = SL(2, \mathbb{F}_{2^n})$ has more or less subgroups that can be exploited more or less easily to find collisions.

The attack proposed by Steinwandt et al. [251] focuses on composite numbers n, let us say $n = n_1n_2$, and polynomials $P_n(X)$ admitting a functional decomposition $P_n(X) = P_{n_1}(P_{n_2}(X))$. The attack exhaustively tests all bit sequences of length n_2 until one is found that hashes to a matrix M with trace $Y := P_{n_2}(X)$. The bit sequence is repeated $\mathrm{ord}(M)$ times to produce a collision.

As $n_2 < n$ no polynomial reduction is done and the matrices have the form given by Proposition 5.5, hence the trace has degree n_2. There are 2^{n_2} polynomials of degree n_2 and also 2^{n_2} bit sequences so this step will *a priori* succeed with a good (non negligible) probability. Moreover as $P_{n_1}(Y) = 0 \bmod P_n(X)$, the trace Y belongs to $\mathbb{F}_{2^{n_1}} \subset \mathbb{F}_{2^n}$ so according to Proposition 5.3, the matrix M has order smaller than $2^{n_1}+1$. The total size of the collision is $n_2.\text{ord}(M) \leq n_2(2^{n_1}+1)$ and the computational cost of the attack is about 2^{n_2}. As an example for $n = 147$ and $P_n(X) = X^{147} + X^{98} + 1 = (x^{49})^3 + (X^{49})^2 + 1$, the attacker may choose $n_1 = 3$ and $n_2 = 49$ to produce collisions of size smaller than 441 bits in time 2^{49}.

When $n = n_1 n_2$ but $P_n(X)$ does not admit a functional decomposition $P_n(X) = P_{n_1}(P_{n_2}(X))$, the subgroup attack might still be possible but it seems much harder. In the beginning of the attack we need a matrix $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ with trace in $\mathbb{F}_{2^{n_1}}$. There are about 2^{2n+n_1} such matrices (about 2^n choices for a and b, then about 2^{n_1} choices for d and one choice for c) while there are about 2^{3n} matrices in $SL(2, \mathbb{F}_{2^n})$, so the probability to reach a matrix of the correct form by random methods is about $\frac{1}{2^{n-n_1}}$. The computational cost of the attack is 2^{n-n_1} and the collision size is bounded by $(n-n_1)2^{n_1}$. In particular, choosing n_1 as the largest proper divisor of n accelerates the attack but produces larger collisions.

The attack is less efficient when the polynomial is not decomposable because there is no more guaranty that any element of degree smaller than n_2 belongs to $\mathbb{F}_{2^n}$. The probability for such an event to happen on random polynomials is *a priori* quite weak. Curiously, Steinwandt et al. give an example with $n = 140$ and $n_2 = 10$, some non-decomposable polynomial and a bit sequence of length 16 hashing to a matrix whose trace belongs to $\mathbb{F}_{2^{10}}$. The authors do not give more details on how the polynomial was chosen or the bit sequence found. According to our analysis, the probability that a bit sequence of size 16 hashes to an element with trace in $\mathbb{F}_{2^{10}}$ is very small, about $2^{16}2^{14-140} = 2^{-110}$. We suspect that this collision was created with the trapdoor attack (Section 5.3.5) but presented with the subgroup attack to illustrate the fact that in some cases subgroup attacks are possible even with non-decomposable polynomials.

5.3.6 Other subgroup attacks

The previous section showed particularly efficient subgroup attacks for composite n; when n is prime, Camion's attack (Section 4.2.6) and improvements upon it also give some satisfactory result, although more limited.

Camion's attack requires a subgroup tower sequence $G = G_0 \supset G_1 \supset G_2 \supset ... \supset G_N = \{I\}$ and its complexity is about $\sqrt{B}$ where $|G_{i-1}|/|G_i| \leq B$ for all i. According to Proposition 5.1, it seems natural to choose for G_N one of the largest Abelian subgroups of $SL(2, \mathbb{F}_{2^n})$ that has order $2^n \pm 1$. Abelian subgroups C_{2^n+1} of order 2^n+1 leads to the sequence $G \supset C_{2^n+1} \supset \{I\}$ for which $B \approx 2^{2n}$ while Abelian subgroups of order 2^n-1 are more interesting since they lead to a sequence $G \supset G_1 \supset G_2 \supset \{I\}$ with $B \approx 2^n$.

Steinwandt et al. recommend choosing n such that both 2^n+1 and 2^n-1 have large factors "in order to make the search for elements of small order not unnecessarily easy"[251]. At the light of the above analysis, this condition is not necessary: the complexity of subgroup attacks is determined equally or more by the difficulty to "reach G_1 from G" than by the difficulty to "reach the identity from G_N". Unless a significantly new idea appears to reach the first subgroup, the complexity of any subgroup attack when n is prime will be $2^{n/2}$ no matter what the factorizations of 2^n+1 and 2^n-1 are.

Let us now explicit Camion's attack by choosing the following chain of subgroups

$$\begin{aligned} G_1 &= \mathcal{T}^{up} \\ G_2 &= \mathcal{T}_1^{up} \\ G_3 &= \{I\}, \end{aligned}$$

where $\mathcal{T}^{up}$ is the upper triangular subgroup of $SL(2,\mathbb{F}_{2^n})$ and $\mathcal{T}_1^{up}$ is the Abelian group of order 2^n defined by

$$\mathcal{T}_1^{up} := \left\{ \left(\begin{smallmatrix} 1 & b \\ 0 & 1 \end{smallmatrix}\right) \middle| \, b \in \mathbb{F}_{2^n} \right\}.$$

The lengths in Camion's attack can be chosen as

$$\begin{aligned} l_1 &= \frac{n}{2} \\ l_2 &= 2^{n/4} \\ l_3 &= 1. \end{aligned}$$

The first value l_1 comes from the fact that random products of A_0 and A_1 basically yield random cosets in $\mathcal{T}^{up}$. The second value l_2 must be large enough so that products of length l_2 of two random elements in an Abelian group of order 2^n-1 yield $\sqrt{2^n}$ different elements. The third length is 1 because the elements of G_2 have order 2. We obtain factorizations of the identity of size $4n2^{n/4}$ with a computational cost of order $2^{n/2}$. The size of the factorization can be reduced a lot by obtaining more than two different products which belong to G_1. Moreover, the time complexity of the second step can be improved drastically by solving discrete logarithms, as we now elaborate.

5.4 New collision and preimage attacks

In this section, we decompose the representation problem into its hard and easy components and we exploit this decomposition to build new collision and preimage attacks against the Zémor-Tillich hash function. Our collision attack improves upon Camion's "tower-of-subgroups" attack as it considerably reduces the time complexity of its second step. The preimage attack extends the collision attack with (interestingly) the same time complexity. Our attacks are generic in the sense that they do not depend on the parameters; in particular they work even if n is prime. With a time complexity close to $2^{n/2}$, they beat by far the birthday bound and ideal preimage complexities which are $2^{3n/2}$ and 2^{3n} for the Zémor-Tillich hash function.

In Section 5.4.1 we present our results on the representation problem; in Sections 5.4.2 and 5.4.3 we give collision and preimage attacks of time complexity close to $2^{n/2}$ but large memory requirements, and in Section 5.4.4 we remove the memory requirements using distinguished points techniques.

5.4.1 Hard and easy components of collision search

In this section, we consider the generic subgroups of $SL(2,\mathbb{F}_{2^n})$ (subgroups existing for any parameter n), including the subgroups of diagonal or triangular matrices and the subgroups of matrices with a given left or right eigenvector. We show that finding elements of these subgroups together with their factorization is nearly as hard as finding collisions for the Zémor-Tillich hash function. As our reductions involve solving discrete logarithms in $\mathbb{F}_{2^n}^*$ we do not claim PPT (probabilistic polynomial time) reductions but reductions that are practical for the parameters initially suggested by Zémor and Tillich.

We start with an easy proposition that will simplify our proofs later.

Proposition 5.8

(a) Let $\begin{pmatrix} a & b \end{pmatrix}, \begin{pmatrix} a' & b' \end{pmatrix} \in \mathbb{F}_{2^n}^2$ *with* $a, a' \neq 0$ *and* $M \in SL(2,\mathbb{F}_{2^n})$ *such that* $\begin{pmatrix} a & b \end{pmatrix} M = \begin{pmatrix} a' & b' \end{pmatrix}$. *Then there exists* $\epsilon \in \mathbb{F}_{2^n}$ *such that* $M = \begin{pmatrix} a^{-1} & b \\ 0 & a \end{pmatrix} \begin{pmatrix} a' & b' \\ 0 & a'^{-1} \end{pmatrix} + \epsilon \begin{pmatrix} b \\ a \end{pmatrix} \begin{pmatrix} a' & b' \end{pmatrix}$.

(b) If $M_1 = \begin{pmatrix} a_0^{-1} & b_0 \\ 0 & a_0 \end{pmatrix} \begin{pmatrix} a_1 & b_1 \\ 0 & a_1^{-1} \end{pmatrix} + \epsilon_1 \begin{pmatrix} b_0 \\ a_0 \end{pmatrix} \begin{pmatrix} a_1 & b_1 \end{pmatrix}$ *and* $M_2 = \begin{pmatrix} a_1^{-1} & b_1 \\ 0 & a_1 \end{pmatrix} \begin{pmatrix} a_2 & b_2 \\ 0 & a_2^{-1} \end{pmatrix} + \epsilon_2 \begin{pmatrix} b_1 \\ a_1 \end{pmatrix} \begin{pmatrix} a_2 & b_2 \end{pmatrix}$ *then* $M_1 M_2 = \begin{pmatrix} a_0^{-1} & b_0 \\ 0 & a_0 \end{pmatrix} \begin{pmatrix} a_2 & b_2 \\ 0 & a_2^{-1} \end{pmatrix} + (\epsilon_1 + \epsilon_2) \begin{pmatrix} b_0 \\ a_0 \end{pmatrix} \begin{pmatrix} a_2 & b_2 \end{pmatrix}$.

Proof: Part (a) is implied by the two following observations:

- For $\epsilon = 0$ we have $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)\left(\begin{smallmatrix} a^{-1} & b \\ 0 & a \end{smallmatrix}\right)\left(\begin{smallmatrix} a' & b' \\ 0 & a'^{-1} \end{smallmatrix}\right) = \left(\begin{smallmatrix} a' & b' \end{smallmatrix}\right)$.

- If $M_1, M_2 \in SL(2, \mathbb{F}_{2^n})$ satisfy $(a,b)M_1 = (a,b)M_2 = (a',b')$ then $M_1 + M_2 = \epsilon\left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right)\left(\begin{smallmatrix} a' & b' \end{smallmatrix}\right)$. Indeed, let c, d such that $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ is unimodular and let $\left(\begin{smallmatrix} a' & b' \\ c_1 & d_1 \end{smallmatrix}\right) := \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) M_1$ and $\left(\begin{smallmatrix} a' & b' \\ c_2 & d_2 \end{smallmatrix}\right) := \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) M_2$. As M_1, M_2 and $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ are in $SL(2, \mathbb{F}_{2^n})$, we have $\det\left(\begin{smallmatrix} a' & b' \\ c_1 & d_1 \end{smallmatrix}\right) = \det\left(\begin{smallmatrix} a' & b' \\ c_2 & d_2 \end{smallmatrix}\right) = 1$. We get

$$\begin{aligned} M_1 + M_2 &= \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)^{-1}\left[\left(\begin{smallmatrix} a' & b' \\ c_1 & d_1 \end{smallmatrix}\right) + \left(\begin{smallmatrix} a' & b' \\ c_2 & d_2 \end{smallmatrix}\right)\right] = \left(\begin{smallmatrix} d & b \\ c & a \end{smallmatrix}\right)\left(\begin{smallmatrix} 0 & 0 \\ c_1+c_2 & d_1+d_2 \end{smallmatrix}\right) \\ &= \left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right)\left(\begin{smallmatrix} c_1+c_2 & d_1+d_2 \end{smallmatrix}\right). \end{aligned}$$

  Moreover, as $\left(\begin{smallmatrix} c_1+c_2 & d_1+d_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right) = a(d_1+d_2) + b(c_1+c_2) = (ad_2 + bc_2) + (ad_1 + bc_1) = 0$, we get the result.

Part (b) is a straightforward computation. □

We now define the short (generalized) representation problem in $\mathbb{F}_{2^n}^*$ and we show how it can be solved for small n (and certainly if $n \leq 170$). The short representation problem is Problem 3.3 with an additional size constraint.

Problem 5.1 Short representation problem in $\mathbb{F}_{2^n}^*$: *Given N (randomly chosen) elements $g_i \in \mathbb{F}_{2^n}^*$, find a factorization $\prod g_i^{e_i} = 1$ such that $\sum |e_i|$ is* not *too large.*
Short generalized representation problem in $\mathbb{F}_{2^n}^*$: *Given N (randomly chosen) elements $g_i \in \mathbb{F}_{2^n}^*$ and a (randomly chosen) element $g_0 \in \mathbb{F}_{2^n}^*$, find a factorization $\prod g_i^{e_i} = g_0$ such that $\sum |e_i|$ is* not too large.

Proposition 5.9 *The (generalized) representation problem can be solved in groups $\mathbb{F}_{2^n}^*$ where the discrete logarithm problem can be solved.*

Proof: Let $g_i \in \mathbb{F}_{2^n}^*, i = 0, ...N$. Let g a generator of $\mathbb{F}_{2^n}^*$, and let α_i be the discrete logarithms of g_i with respect to base g. The representation problem amounts to solving the following problem: find $\{e_i\}$ such that $\sum e_i \alpha_i = \alpha_0 \bmod (2^n - 1)$ and $\sum |e_i|$ is *not too large.* A good solution to this problem can be computed with the LLL algorithm [165].□

If the exponents α_i are random numbers uniformly distributed in $[1, 2^n - 1]$ the smallest solution has expected size $\sum_i |e_i|$ about $N 2^{n/N}$ (approximating that there is no collision, the sums $\sum e_i \alpha_i$ for $e_i \leq 2^{n/N}$ produce the $2^n - 1$ possible values). The LLL algorithm actually gives a solution such that $\sum |e_i|^2$ is close to optimal, but this is enough for our purposes. By the LLL approximation bound, the solution provided using LLL has a norm 2 smaller than $\sqrt{N} 2^{n/N+N}$ which is subexponential for $N \approx \sqrt{n}$. In practice, LLL performs much better and in the analysis of our algorithms, we will approximate that the size of the solution given by LLL algorithm is also about $N2^{n/N}$.

With this method, the representation problem in $\mathbb{F}_{2^n}^*$ can be solved if discrete logarithms can be computed, in particular the representation problem can be solved today for $n \leq 170$. The following result follows from Proposition 5.9.

Proposition 5.10 *Let n be such that discrete logarithms can be solved in $\mathbb{F}_{2^n}^*$. Let $\mathcal{D}, \mathcal{T}^{up}, \mathcal{T}^{low}, \mathcal{L}^v, \mathcal{R}^v \subset SL(2, \mathbb{F}_{2^n})$ be the subgroups of diagonal, upper and lower triangular matrices and the subgroup of matrices with left or right eigenvector v. If an attacker can compute N random elements M_i of one of these subgroups together with bit sequences m_i of length at most L hashing to these matrices, then he can also find a message m such that $h_{ZT}(m) = I$. The message m has expected size smaller than $NL2^{n/N}$ in the diagonal case and smaller than $NL2^{1+n/N}$ in the other cases.*

Proof: We use the descriptions of subgroup elements in Proposition 5.4. Any diagonal matrix can be written as $D_i = \begin{pmatrix} a_i & 0 \\ 0 & a_i^{-1} \end{pmatrix}$ for some $a_i \in \mathbb{F}_{2^n}^*$. Let $\{e_i\}$ be a solution to the representation problem with respect to $\{a_i\}$, that is $\prod a_i^{e_i} = 1$. Construct m as the concatenation of e_1 messages m_1, e_2 messages m_2, etc. (in any order). Then $H_{ZT}(m) = \prod D_i^{e_i} = \begin{pmatrix} \prod a_i^{e_i} & 0 \\ 0 & \prod a_i^{-e_i} \end{pmatrix} = I$.

Similarly, an upper triangular matrix T_i can be written as $\begin{pmatrix} a_i & b_i \\ 0 & a_i^{-1} \end{pmatrix}$ for some $a_i \in \mathbb{F}_{2^n}^*, b_i \in \mathbb{F}_{2^n}$. Let $\{e_i\}$ be a solution to the representation problem with respect to $\{a_i\}$, that is $\prod a_i^{e_i} = 1$. Construct m' as the concatenation of e_1 messages m_1, e_2 messages m_2, etc. (in any order) and $m = m'||m'$. Then $H_{ZT}(m') = \begin{pmatrix} 1 & b \\ 0 & 1^{-1} \end{pmatrix}$ for some $b \in \mathbb{F}_{2^n}$ and $H_{ZT}(m) = I$.

By definition each $M_i \in \mathcal{L}^{(a\ b)}$ satisfies $(a\ b) M_i = \lambda_i (a\ b)$ for some $\lambda_i \in \mathbb{F}_{2^n}^*$. Let $\{e_i\}$ be a solution to the representation problem with respect to $\{\lambda_i\}$, that is $\prod \lambda_i^{e_i} = 1$. Construct m' as the concatenation of e_1 messages m_1, e_2 messages m_2, etc. (in any order) and $m = m'||m'$. Then $(a\ b) H_{ZT}(m') = (a\ b)$ which by Proposition 5.8 implies $H_{ZT}(m') = I + \epsilon \begin{pmatrix} b \\ a \end{pmatrix} (a\ b)$ hence $H_{ZT}(m) = I$.

The proof for $\mathcal{T}^{low}$ and $\mathcal{R}^v$ are similar and the claim on the message lengths follows from our analysis of the representation problem in $\mathbb{F}_{2^n}^*$. □

The parts of Proposition 5.10 concerning $\mathcal{L}^v$ and $\mathcal{R}^v$ have interesting graph interpretations that we give for $\mathcal{L}^v$ in Section 5.5.3.

5.4.2 A new generic collision attack

We now give an algorithm finding N_2 matrices M_i such that $\begin{pmatrix} 1 & 0 \end{pmatrix} M_i = \lambda_i \begin{pmatrix} 1 & 0 \end{pmatrix}$ for some $\lambda_i \in \mathbb{F}_{2^n}^*$, and combining them as in Proposition 5.10 to find collisions for the Zémor-Tillich hash function.

We denote by $\mathbb{P}^1(\mathbb{F}_{2^n})$ the projective space of dimension 1 on $\mathbb{F}_{2^n}$, which is the set of equivalence classes of $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ that results from identifying two vectors $\begin{pmatrix} a_1 & b_1 \end{pmatrix}$ and $\begin{pmatrix} a_2 & b_2 \end{pmatrix}$ if and only if $\begin{pmatrix} a_2 & b_2 \end{pmatrix} = \lambda \begin{pmatrix} a_1 & b_1 \end{pmatrix}$ for some $\lambda \in \mathbb{F}_{2^n}^*$. We denote by $[a : b]$ the projective point that is the equivalence class of a vector $\begin{pmatrix} a & b \end{pmatrix}$. To any message $m = m_1 m_2 ... m_k$ we associate two projective points $q(m), q_{-1}(m) \in \mathbb{P}^1(\mathbb{F}_{2^n})$ as follows. We define

$$\begin{aligned} \begin{pmatrix} a(m) & b(m) \end{pmatrix} &:= \begin{pmatrix} 1 & 0 \end{pmatrix} \prod_{i=1}^{k} M_{m_i} = \begin{pmatrix} 1 & 0 \end{pmatrix} H_{ZT}(m), \\ \begin{pmatrix} a'(m) & b'(m) \end{pmatrix} &:= \begin{pmatrix} 1 & 0 \end{pmatrix} \prod_{i=k}^{1} M_{m_i}^{-1} = \begin{pmatrix} 1 & 0 \end{pmatrix} H_{ZT}(m)^{-1}, \end{aligned}$$

then $q(m) := [a(m) : b(m)]$ and $q_{-1}(m) := [a'(m) : b'(m)]$.

Our algorithm first performs a birthday attack [273] to find collisions on the q values as follows. Random messages m and m' of size $k > n/2$ are generated and stored together with $q(m)$ and $q_{-1}(m')$, until m_1, m_2 are found such that $q(m_1) = q_{-1}(m_2)$ (see Figure 5.2). As there are $2^n + 1$ points in $\mathbb{P}^1(\mathbb{F}_{2^n})$, the probability that $q(m_1) = q_{-1}(m_2)$ for some m_1, m_2 is $1 - \left(1 - \frac{2^{N_1}}{2^n+1}\right)^{2^{N_1}}$ after 2^{N_1} steps. In particular, after $2^{N_1} = 2^{n/2}$ steps we have a probability $1 - e^{-1} \approx 0.63$ to know a message $m := m_1 || m_2$ of size $2k$ such that $\begin{pmatrix} 1 & 0 \end{pmatrix} h_{ZT}(m) = \lambda \begin{pmatrix} 1 & 0 \end{pmatrix}$ for some $\lambda \in \mathbb{F}_{2^n}^*$.

This collision search is repeated until N_2 distinct messages m_i are found such that $\begin{pmatrix} 1 & 0 \end{pmatrix} h_{ZT}(m_i) = \lambda_i \begin{pmatrix} 1 & 0 \end{pmatrix}$ for some $\lambda_i \in \mathbb{F}_{2^n}^*$. To guarantee that the collisions found are all distinct, we may perform each collision search with a different length $k > n/2$, or choose k slightly larger than $n/2 + \log_2(N_2)$, say $k = n/2 + \log_2(N_2) + 10$.

The next step of the algorithm combines the messages m_i to get a collision for the Zémor-Tillich hash function. As in the proof of Proposition 5.10, we compute a solution $\{e_i\}$ to the representation problem in $\mathbb{F}_{2^n}^*$ with respect to the λ_i, that is $\prod \lambda_i^{e_i} = 1$. From this solution, we finally construct a message m' as the concatenation of each message m_i repeated e_i times (in any order), and a message $m = m' || m'$ that collides with the void message as shown in the proof of Proposition 5.10.

To analyze this attack, suppose that the N_2 collision searches are done with $k = n/2 + 1, ..., n/2 + N_2$ and that the algorithm described in Sec-

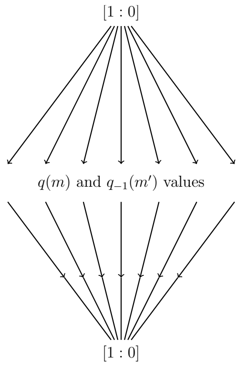

Figure 5.2: Collision search on q values.

tion 5.4.1 is used to solve the representation problem. The expected size of the collision is then bounded by $(n/2 + N_2)N_2 2^{n/N_2+2}$, the memory requirement is $2^{n/2+1}n$ and the time complexity is $N_2 2^{n/2+1}t + t_{REP}$ where t is the time needed to compute one q value and t_{REP} is the time needed to solve the representation problem. In particular for $n = 130$ and $N_2 = 16$, this attack produces a collision to the void message of size about 2^{18} in time $2^{69}t$ and memory requirements 2^{69}. The memory requirements will be removed in Section 5.4.4 by using distinguished points techniques [219].

5.4.3 A new generic preimage attack

We now extend our ideas to a preimage attack. Interestingly, this attack has essentially the same complexity as the collision attack.

Suppose we want to find a preimage to a matrix $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, that is a message $m = m_1...m_k$ such that $M = H_{ZT}(m) = \prod M_{m_i}$. As we showed in previous section, random messages m_i of size $L > n$ such that $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}(m_i) = \lambda_i \left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right)$ for some $\lambda_i \in \mathbb{F}_{2^n}^*$ can be found with memory $n2^{n/2+1}$ and time $2^{n/2+1}t$. Similarly, random messages $m_i, i = 0, ...N_2$ of size $L > n$ satisfying $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}(m_0) = \lambda_0 \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) H_{ZT}(m_i) = \lambda_i \left(\begin{smallmatrix} a & b \end{smallmatrix}\right), i > 0$ for some $\lambda_i \in \mathbb{F}_{2^n}^*$ can also be found with the same time and memory complexities.

Solving a (generalized) representation problem, we can compute $\{e_i\}$ such that $\prod \lambda_i^{e_i} = \lambda_0$, hence we can compute a message m'_0 of size $N_2 L 2^{n/N_2}$ and a matrix $M_0 := H_{ZT}(m'_0)$ such that $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) M_0 = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$. Similarly, from N_3 different solutions to the representation problem $\prod \lambda_i^{e_i} = 1$ we get N_3 messages m'_i of size $N_2 L 2^{n/N_2}$ such that $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) H_{ZT}(m'_i) = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$. Let $\left(\begin{smallmatrix} c' & d' \end{smallmatrix}\right) := \left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right) H_{ZT}(m'_0)$. As $ad' + bc' = ad + bc = 1$, we have $a(d + d') + b(c + c') = 0$, that is $\left(\begin{smallmatrix} c+c' & d+d' \end{smallmatrix}\right) = \delta_0 \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ for some $\delta_0 \in \mathbb{F}_{2^n}$.

According to Proposition 5.8, for all $i > 0$ there exists $\delta_i \in \mathbb{F}_{2^n}$ such that $H_{ZT}(m'_i) = \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) + \delta_i \left(\begin{smallmatrix}b\\a\end{smallmatrix}\right)\left(\begin{smallmatrix}a&b\end{smallmatrix}\right)$; moreover we have $H_{ZT}(m'_{i_1})H_{ZT}(m'_{i_2}) = \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) + (\delta_{i_1} + \delta_{i_2})\left(\begin{smallmatrix}b\\a\end{smallmatrix}\right)\left(\begin{smallmatrix}a&b\end{smallmatrix}\right)$. Suppose the δ_i values generate $\mathbb{F}_{2^n}/\mathbb{F}_2$, which is very likely if N_3 is shortly bigger than n, say $N_3 = n + 10$. Then by solving a binary linear system, we can write $\delta_0 = \sum_{i\in I}\delta_i$ for some $I \subset \{1, ..., N_3\}$ of size $\leq n$ and hence $M_1 := \prod_{i\in I} H_{ZT}(m'_i) = \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) + \delta_0\left(\begin{smallmatrix}b\\a\end{smallmatrix}\right)\left(\begin{smallmatrix}a&b\end{smallmatrix}\right)$. Finally, we have $M_0M_1 = \left(\begin{smallmatrix}a&b\\c'&d'\end{smallmatrix}\right)\left[\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) + \delta_0\left(\begin{smallmatrix}b\\a\end{smallmatrix}\right)\left(\begin{smallmatrix}a&b\end{smallmatrix}\right)\right] = \left(\begin{smallmatrix}a&b\\c&d\end{smallmatrix}\right)$.

This shows that any message made of m'_0 concatenated with any concatenation of the messages $m'_i, i \in I$, is a preimage to $\left(\begin{smallmatrix}a&b\\c&d\end{smallmatrix}\right)$. The collision size is about bounded by $N_3(n/2 + N_2)N_2 2^{n/N_2+2}$, that is $12n^2(n + 10)$ if $N_2 = n$ and $N_3 = n + 10$. The memory requirement of this attack is $2^{n/2+1}n$ and the time complexity is $N_2 2^{n/2+1}t + t_{REP}$ where t is the time needed to compute one q value and t_{REP} is the time needed to solve the representation problem (note that finding N_3 solutions to a representation problem essentially requires the same time as finding one solution because both times are essentially determined by the computation of the discrete logarithms). As for our collision attack, the memory requirements can be removed by using distinguished points techniques.

5.4.4 Memory-free versions of our attacks

The attacks of Sections 5.4.2 and 5.4.3 require storing two databases of about $2^{n/2}$ projective points in $\mathbb{P}^1(\mathbb{F}_{2^n})$ and their corresponding messages. We now remove the memory requirements by using distinguished points techniques [219].

Let $\alpha : \mathbb{P}^1(\mathbb{F}_{2^n}) \to \{0,1\}^k$ and $\beta : \mathbb{P}^1(\mathbb{F}_{2^n}) \to \{0,1\}$ be two "pseudorandom functions" and let $\varphi : \mathbb{P}^1(\mathbb{F}_{2^n}) \to \mathbb{P}^1(\mathbb{F}_{2^n})$ be defined by

$$p \to \varphi(p) = \begin{cases} q(\alpha(p)) & \text{if } \beta(p) = 0 \\ q_{-1}(\alpha(p)) & \text{if } \beta(p) = 1, \end{cases}$$

where $k > n$ is arbitrarily chosen and q and q_{-1} are defined as in Section 5.4.2.

The iterates $q_0, \varphi(q_0), \varphi(\varphi(q_0)), ...$ of φ on q_0 all belong to the finite domain $\mathbb{P}^1(\mathbb{F}_{2^n})$ so at some point iterating φ will produce a collision (see Figure 5.3), that is two points p_1 and p_2 such that $\varphi(p_1) = \varphi(p_2) = c$. If the behavior of φ is sufficiently random then $\beta(p_1) \neq \beta(p_2)$ with a probability 1/2, in which case $\alpha(p_1)$ and $\alpha(p_2)$ can be combined to produce a message m of size $2k$ such that $\left(\begin{smallmatrix}1&0\end{smallmatrix}\right) h_{ZT}(m) = \lambda \left(\begin{smallmatrix}1&0\end{smallmatrix}\right)$ for some $\lambda \in \mathbb{F}^*_{2^n}$.

The functions α and β do not need to be "pseudorandom" in the strong cryptographic meaning, but only "sufficiently pseudorandom" for the above analysis to hold.

Now that the problem of finding a collision on the q values has been translated in the problem of detecting a cycle in the iterates of φ, we can remove the memory requirements by standard techniques. We recall here the method of *distinguished points*; other methods are described in [242]. Let $\mathcal{D}_d := \{q = [a:b] \in \mathbb{P}^1(\mathbb{F}_{2^n}) | b \neq 0, lsb_d(a/b) = 0^d\}$ be sets of 2^{n-d} distinguished q values such that their d last bits are all 0. During the collision search, we only store the q values that belong to $\mathcal{D}$ and only look for collisions on these particular q values. Finding a collision c' on distinguished points requires 2^{d-1} additional steps in average but the memory is reduced to $2^{n/2-d}$; if $d = n/2 - 10$ the time overhead is negligible and the memory requirements are very small (see Figure 5.4).

From the two distinguished points p_1' and p_2' that precede c' in the iterates of φ, we can recover the points p_1 and p_2 that produce the actual collision c as follows. Iterate again φ on p_1' and p_2' and store only distinguished points but this time with $d = n/2 - 20$. After about $2^{n/2-10}$ steps on each side (and a small memory of about 2^{11}) a collision c'' and preceding distinguished points p_1'' and p_2'' are found that are closer to the actual collision c, p_1, p_2. Iterating again from p_1'' and p_2'' with a larger distinguished-point set, we finally get the actual collision with small time overhead and small memory.

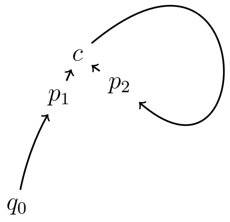

Figure 5.3: Iterating φ from some initial point q_0, we eventually get a collision c

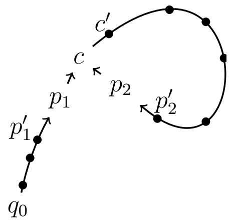

Figure 5.4: Collision graph with markers on the distinguished points. The average distance between two distinguished points is 2^d. The average length of the path is $2^{n/2}$. Finding a collision on a distinguished point requires essentially the same time as finding a general collision, as soon as $2^d << 2^{n/2}$.

With this method instead of the trivial collision search steps, our collision and preimage attacks require negligible memory and essentially the same time complexity. As the output of Zémor-Tillich is about $3n$ bits, these attacks are far better than birthday and optimal preimage bounds. In the following sections, we introduce two variants of Zémor-Tillich with reduced output sizes respectively $2n$ and n bits, and we show that these variants are essentially as secure as the original Zémor-Tillich for sufficiently small parameters including the parameters initially suggested in [258].

5.5 New variants of ZT hash

The attacks of Section 5.4 suggest that the output of the Zémor-Tillich hash function should be of n bits rather than $3n$ bits. In this section, we introduce two variants of ZT hash, the vectorial and the projective variants with output sizes respectively $2n$ and n bits. We show that the original function is collision resistant if and only if its vectorial variant and (for small n) if and only if its projective variant are collision resistant. Section 5.5.1 discusses the vectorial variant and Section 5.5.2 the projective variant. Graphical interpretations of our results are given in Section 5.5.3.

5.5.1 Vectorial variant of Zémor-Tillich

Our first variant H_{ZT}^{vec} is simply the first row of Zémor-Tillich, that is

$$H_{ZT}^{vec}\left(P_n(X), m\right) := \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$$

if $H_{ZT}\left(P_n(X), m\right) = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$. Alternatively, we may parameterize the function H_{ZT}^{vec} by an initial vector $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \neq \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)$ as $H_{ZT}^{vec}\left(P_n(X) || \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) := \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right)$. Clearly, the output has $2n$ bits.

As for the original function, the security of this variant can be related to both algebraic and graph problems. Finding a collision corresponds to finding two messages m and m' such that $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right) = \left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m'\right)$, in particular it is enough to find one message m such that $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right) = \left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right)$. Finding a preimage to a vector $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ is finding a message m such that $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right) = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$.

The graph associated to this function is the H_{ZT}^{vec} graph described in Section 5.5.3 (see Figure 5.5). Finding a collision for H_{ZT}^{vec} is finding two paths in this graph starting and ending at the same vertex; in particular, finding a cycle in H_{ZT}^{vec} is enough to find a collision for H_{ZT}^{vec}. Computing a preimage to a vertex $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ is finding a path from $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$ to $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$.

The following proposition shows that H_{ZT}^{vec} is collision resistant if and only if the original function H_{ZT} is collision resistant.

Proposition 5.11 *If there exists a* PPT *(probabilistic polynomial time) algorithm that for randomly chosen starting vectors* $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \neq \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)$ *finds a collision on* $H_{ZT}^{vec}\left(P_n(X) || \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), .\right)$*, then there exists a* PPT *algorithm finding collisions for the original Zémor-Tillich function with the same polynomial as parameter.*

PROOF: Given a PPT algorithm A^{vec} finding collisions for the vectorial version, we build a PPT algorithm A^{mat} finding collisions for the original matrix version. The algorithm A^{mat} first picks a random matrix $M_0 := \left(\begin{smallmatrix} a_0 & b_0 \\ c_0 & d_0 \end{smallmatrix}\right) \in$

$SL(2,\mathbb{F}_{2^n})$ and runs A^{vec} on (a_0,b_0) to get two messages m_{10} and m_{11} corresponding to matrices M_{10} and M_{11} such that $(a_0,b_0)M_{10}=(a_0,b_0)M_{11}=(a_1,b_1)$. Without loss of generality, we can assume that (a_1,b_1) is randomly uniformly distributed, otherwise we may just append the same randomly chosen sequence of bits to both messages. Similarly, we can also assume that $a_1\neq 0$ since adding the same single bit to both paths would remove this assumption. Algorithm A^{mat} then calls again A^{vec} on (a_1,b_1) to get two matrices M_{20} and M_{21}, etc. It repeats this operation $n+1$ times.

Let $v_i := \left(\begin{smallmatrix} a_i & b_i\end{smallmatrix}\right)$ and $\widetilde{v_i} := \left(\begin{smallmatrix} b_i \\ a_i\end{smallmatrix}\right)$. According to Proposition 5.8(a), the matrices M_{ij} can be written as

$$M_{ij}=\begin{pmatrix} a_{i-1}^{-1} & b_{i-1} \\ 0 & a_{i-1}\end{pmatrix}\begin{pmatrix} a_i & b_i \\ 0 & a_i^{-1}\end{pmatrix}+\epsilon_{ij}\widetilde{v_{i-1}}v_i$$

for some $\epsilon_{ij}\in\mathbb{F}_{2^n}$. Applying Proposition 5.8(b) recursively, for any $e=e_1...e_{n+1}\in\{0,1\}^{n+1}$, we have

$$\prod_{i=1}^{n+1}M_{ie_i}=\begin{pmatrix} a_0^{-1} & b_0 \\ 0 & a_0\end{pmatrix}\begin{pmatrix} a_{n+1} & b_{n+1} \\ 0 & a_{n+1}^{-1}\end{pmatrix}+\left(\sum_{i=1}^{n+1}\epsilon_{ie_i}\right)\widetilde{v_0}v_{n+1}.$$

For $1\leq i\leq n+1$, let $\epsilon_i := \epsilon_{i0}+\epsilon_{i1}$. Seeing each ϵ_i as a binary vector of length n over $\mathbb{F}_2$, these vectors are linearly dependent. Moreover, finding a subset I of $\{1,...,n+1\}$ such that $\sum_{i\in I}\epsilon_i=0$ simply amounts to invert a binary linear system, which is cubic in $n+1$.

We now conclude the description of A^{mat}. After computing $I\subset\{1,...,n+1\}$ such that $\sum_{i\in I}\epsilon_i=0$, the algorithm A^{mat} returns $m=m_{10}||m_{20}||...||m_{n+1,0}$ and $m'=m_{1e_1}||m_{2e_2}||...||m_{n+1,e_{n+1}}$ where $e_i=1$ if and only if $i\in I$. By the discussion above, it is clear that

$$\begin{aligned} H_{ZT}\left(P_n(X),m'\right) &= H_{ZT}\left(P_n(X),m\right) \\ &= \begin{pmatrix} a_0^{-1} & b_0 \\ 0 & a_0\end{pmatrix}\begin{pmatrix} a_{n+1} & b_{n+1} \\ 0 & a_{n+1}^{-1}\end{pmatrix}+\left(\sum_{i=1}^{n+1}\epsilon_{i0}\right)\widetilde{v_0}v_{n+1}.\end{aligned}$$

□

The reduction of Proposition 5.11 is polynomial but not completely tight: the algorithm A^{mat} runs $n+1$ times the algorithm A^{vec}. Note that if instead of A^{vec} we have an algorithm A'^{vec} returning a message m corresponding to a *cycle* for the vectorial version, then the message $m||m$ is a collision for the matrix version. Indeed, if $\left(\begin{smallmatrix} a & b\end{smallmatrix}\right)M=\left(\begin{smallmatrix} a & b\end{smallmatrix}\right)$ Proposition 5.8(a) shows that M

can be written as $M = \left(\begin{smallmatrix} a^{-1} & b \\ 0 & a \end{smallmatrix}\right)\left(\begin{smallmatrix} a & b \\ 0 & a^{-1} \end{smallmatrix}\right) + \epsilon \left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right)\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) = I + \epsilon \left(\begin{smallmatrix} b \\ a \end{smallmatrix}\right)\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ hence by a straightforward computation $M^2 = I$.

Despite the reduction in Proposition 5.11, the vectorial version of Zémor-Tillich has a weakness that was not present in the original function. For any $a \in \mathbb{F}_{2^n}^*$, we have

$$\left(\begin{smallmatrix} a & aX \end{smallmatrix}\right)\left(\begin{smallmatrix} X & 1 \\ 1 & 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0 & a \end{smallmatrix}\right) = \left(\begin{smallmatrix} a & aX \end{smallmatrix}\right)\left(\begin{smallmatrix} X & X+1 \\ 1 & 1 \end{smallmatrix}\right).$$

Therefore, if the adversary can choose the starting point, it can create a collision $(m, m') := (0, 1)$ by choosing any vector $\left(\begin{smallmatrix} a & aX \end{smallmatrix}\right)$ as starting vector. Alternatively, the adversary can create a collision $(m, m') := (m_1||0||m_2, m_1||1||m_2)$ for any bitstrings m_1, m_2 by "multiplying backward" some vector $\left(\begin{smallmatrix} a & aX \end{smallmatrix}\right)$ by A_0^{-1} and A_1^{-1} according to the bits of m_1. We point out that these weaknesses should not be considered as collision attacks but rather as additional trapdoor attacks specific to the vectorial version. In Section 9.6.2, we will discuss an idea due to Zémor to prevent this weakness.

5.5.2 Projective variant of Zémor-Tillich

Our second variant H_{ZT}^{proj} exploits even further Proposition 5.10. We define

$$H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) := [a : b]$$

where $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) := H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ and $[a : b] \in \mathbb{P}^1(\mathbb{F}_{2^n})$. Finding a collision for $H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), .\right)$ is finding two messages m and m' such that $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right) = \lambda \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m'\right)$ for some λ, in particular it is enough to find a *cyclic* collision which is a message m such that $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$ is a left eigenvector of $H_{ZT}\left(P_n(X), m\right)$.

The output of H_{ZT}^{proj} is very close to n bits. For the parameters suggested by Tillich and Zémor, its collision resistance is equivalent to the collision resistance of the original function.

Proposition 5.12 *If there exists an algorithm that finds collisions on* $H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), .\right)$*, there exists an algorithm that finds collisions on* $H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), .\right)$*, assuming that for some* $n' > n$ *it is feasible to compute* n' *discrete logarithms in* $\mathbb{F}_{2^n}^*$ *and one subset sum problem of size* n'*.*
If we denote by t^{proj}*,* t^{DL} *and* $t^{SS}(n')$ *the times needed respectively to find collisions on the projective version, to solve one discrete logarithm problem in* $\mathbb{F}_{2^n}^*$ *and to solve a subset sum problem of size* n'*, collisions on the vectorial version can be found in time* $n'(t^{proj} + t^{DL}) + t^{KN}(n')$*.*

PROOF: Given an algorithm A^{proj} finding collisions for the projective version, we build an algorithm A^{vec} finding collisions for the vectorial version. Receiving an initial vector $v_0 = (a_0, b_0)$, A^{vec} forwards it to A^{proj} and receives two messages m_{10}, m_{11}. To the two messages correspond two vectors (a_{10}, b_{10}) and $(a_{11}, b_{11}) = \lambda_1(a_{10}, b_{10})$ for some λ_1. The algorithm A^{vec} computes the discrete logarithm d_1 of λ_1 with respect to some generator g of $\mathbb{F}_{2^n}^*$. The algorithm A^{vec} then runs A^{proj} on the projective point (a_{10}, b_{10}) and computes d_2 similarly, etc.

After n' steps, the algorithm A^{vec} computes a subset $I \subset \{1, ..., n'\}$ such that $\sum_{i \in I} d_i = 0 \bmod 2^n - 1$. By concatenating the paths m_{ie_i} where $e_i = 1$ if $i \in I$ and $e_i = 0$ otherwise, algorithm A^{vec} produces a collision with the message $m_{10}||...||m_{n'0}$ for the vectorial version. The output is correct because both messages lead to the vector $\left(\prod_{i \in I} \lambda_i\right)(a_{n'0}, b_{n'0}) = g^{\sum_{i \in I} d_i}(a_{n'0}, b_{n'0}) = (a_{n'0}, b_{n'0})$.

The claim on the running time follows straightforwardly. □

The best choice for n' depends on the exact values of t^{proj}, t^{DL} and $t^{SS}(n')$. Solving discrete logarithms problems is believed to be hard but is definitely feasible in $\mathbb{F}_{2^n}^*$ if $n < 170$. Computing $I \subset \{1, ..., n'\}$ such that $\sum_{i \in I} d_i = 0 \bmod 2^n - 1$ is related to the subset sum problem which is NP-hard but usually easy in average. For the parameters proposed by Zémor-Tillich, lattice reduction algorithms like LLL will probably succeed in performing the reduction. Another method is to use Wagner's "k-lists" algorithm [263] for solving the subset sum problem. This algorithm can solve the subset sum problem in time and space $k2^{n/(1+\log k)}$ which for $k \approx \sqrt{n}$ is roughly $2^{2\sqrt{n}}$ which is about 2^{26} for $n = 170$. The drawback with this method is that n' must also increase to $2^{2\sqrt{n}}$ hence the discrete logarithm costs increase and the quality of the reduction decreases.

Assuming the existence of an algorithm A'^{proj} computing *cyclic* collisions on the projective version (messages m_i such that $(a_0, b_0)H_{ZT}P_n(X)m_i = \lambda_i(a_0, b_0)$ for some λ_i) the reduction slightly improves. Indeed, A^{vec} must only compute a *small integer* solution $(x_1, ..., x_{n'})$ to $\sum_i x_i d_i = 0 \bmod 2^n - 1$ instead of a *binary* solution. The reduction algorithm still has to compute discrete logarithm problems but it must not solve any subset sum problem.

The projective version is subject to the same attacks as the matrix and vectorial versions, plus other trapdoor attacks that were not possible in the matrix version nor even in the vectorial version. With a probability of about one half on the choice of two bitstrings m and m', there exists an initial vector (a_0, b_0) such that $H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) = H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right)$. Indeed, let $\left(\begin{smallmatrix} m_1 & m_2 \\ m_3 & m_4 \end{smallmatrix}\right) := H_{ZT}\left(P_n(X), m\right)$ and $\begin{pmatrix} m'_1 & m'_2 \\ m'_3 & m'_4 \end{pmatrix} := H_{ZT}\left(P_n(X), m'\right)$.

Setting apart the case $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right)$ we can assume without influencing the projective hash result that the initial vector writes as $\left(\begin{smallmatrix} a & 1 \end{smallmatrix}\right)$. The equality of the two projective hash values therefore translates into a quadratic equation in a

$$a^2(m_1m_2'+m_1'm_2)+a(m_1m_4'+m_4'm_1+m_2m_3'+m_2'm_3)+(m_3m_4'+m_3'm_4)=0$$

which has solutions with a probability of about one half.

5.5.3 Graph-theoretical perspectives on our variants

We now provide graphical interpretations of our results. We define the graphs $\mathcal{ZT}^{vec}$ and $\mathcal{ZT}^{proj}$ corresponding to our vectorial and projective versions of Zémor-Tillich, we relate them to the Zémor-Tillich graphs $\mathcal{ZT}$ and we study their properties. Finally, we provide graphical interpretations of our main propositions.

Recall that the Zémor-Tillich hash function is associated to a Cayley graph $\mathcal{ZT}$, in which each vertex corresponds to a matrix $M \in SL(2, \mathbb{F}_{2^n})$ and each edge to a couple $(M_1, M_2) \in SL(2, \mathbb{F}_{2^n})^2$ such that $M_2 = M_1A_0$ or $M_2 = M_1A_1$ [258].

We now construct the graphs $\mathcal{ZT}^{vec}$ and $\mathcal{ZT}^{proj}$ as follows. For $\mathcal{ZT}^{vec}$, we associate a vertex to each row vector $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) \in \mathbb{F}_{2^n}^{1\times 2}\backslash\{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)\}$ and an edge to each couple of such vectors $(\left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right), \left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right))$ satisfying $\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right)A_0$ or $\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right)A_1$. Alternatively, the graph $\mathcal{ZT}^{vec}$ can be constructed from the graph $\mathcal{ZT}$ by identifying two vertices $M_1 = \left(\begin{smallmatrix} a_1 & b_1 \\ c_1 & d_1 \end{smallmatrix}\right)$ and $M_2 = \left(\begin{smallmatrix} a_2 & b_2 \\ c_2 & d_2 \end{smallmatrix}\right)$ when $\left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right)$. An example of graph $\mathcal{ZT}^{vec}$ is shown in Figure 5.5. As shown by the new trapdoor attacks of Section 5.5.1, the $\mathcal{ZT}^{vec}$ graph has girth 1 unlike the original Zémor-Tillich graph.

Similarly, we associate a vertex of $\mathcal{ZT}^{proj}$ to each projective point $q_i = [a_i : b_i] \in \mathbb{P}^1(\mathbb{F}_{2^n})$ and an edge to each couple (q_1, q_2) such that $\lambda\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right)A_0$ or $\lambda\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right)A_1$ for some $\lambda \in \mathbb{F}_{2^n}^*$. Alternatively, the graph $\mathcal{ZT}^{proj}$ may be constructed from the graph $\mathcal{ZT}^{vec}$ by identifying two vertices $\left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right)$ when $\left(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix}\right) = \lambda\left(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}\right)$ for some $\lambda \in \mathbb{F}_{2^n}^*$. The girth of graphs $\mathcal{ZT}^{proj}$ is also 1. Two examples of such graphs are shown in Figure 5.6.

According to Proposition 5.8, finding a cycle in $\mathcal{ZT}^{vec}$ is just as hard as finding a cycle in $\mathcal{ZT}$ because if $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)M = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ then $M^2 = I$. Moreover, according to Proposition 5.11, an algorithm finding two paths in $\mathcal{ZT}^{vec}$ that end at the same vertex and start at a given random vertex is sufficient to find a cycle in $\mathcal{ZT}$.

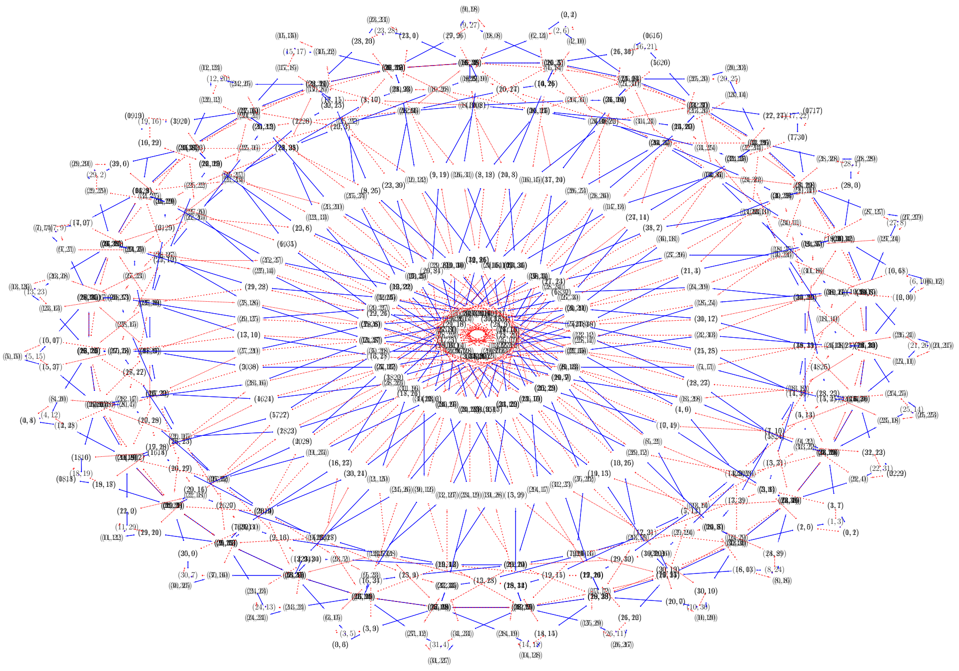

Figure 5.5: $\mathcal{ZT}^{vec}$ graph for parameter $P_5(X) = X^5 + X^2 + 1$. The vertices are labeled by vectors. Red dotted (resp. blue solid) arrows correspond to multiplication by matrix A_0 (resp. A_1). Each polynomial $\sum a_i X^i$ is written as $\sum a_i 2^i$.

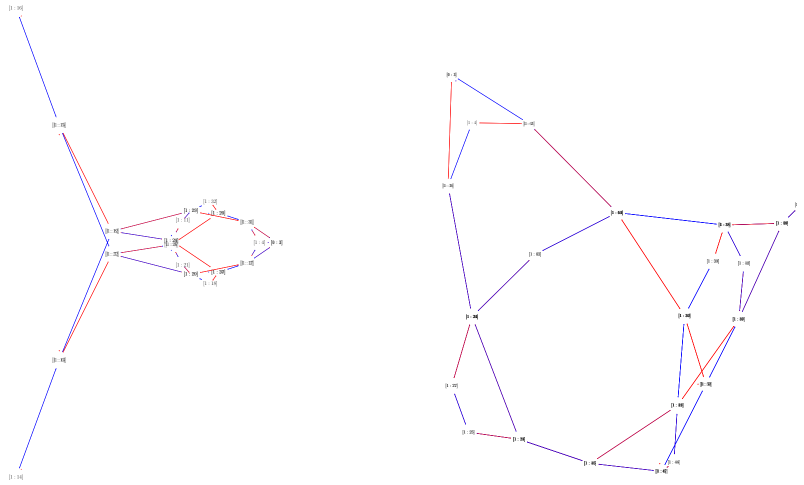

Figure 5.6: $\mathcal{ZT}^{proj}$ graph for parameter $P_5(X) = X^5 + X^2 + 1$ and $P_7(X) = X^6 + X^2 + 1$. The vertices are labeled by projective points. Red (resp. blue) arrows correspond to multiplication by matrix A_0 (resp. A_1). Each polynomial $\sum a_i X^i$ is written as $\sum a_i 2^i$.

The representation of graphs $\mathcal{ZT}^{vec}$ that is given in Figure 5.5 presents a radial symmetry. This symmetry is not surprising as it reflects the relation $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) A_i = \left(\begin{smallmatrix} a' & b' \end{smallmatrix}\right) \Leftrightarrow [\lambda \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)] A_i = [\lambda \left(\begin{smallmatrix} a' & b' \end{smallmatrix}\right)]$: multiplying each vertex of $\mathcal{ZT}^{vec}$ by a constant λ is equivalent to a rotation of the graph. The projective graphs $\mathcal{ZT}^{proj}$ are obtained by identifying the vertices that are equal up to a rotation.

A vertex in the graph $\mathcal{ZT}^{vec}$ can be characterized by a radial and an angular position. A cycle in the graph $\mathcal{ZT}^{proj}$ induces a path in the graph $\mathcal{ZT}^{vec}$ from a vertex to another vertex with the same radial coordinate, but not necessarily the same angular coordinate. Clearly, different such paths can be combined to give a cycle in the graph $\mathcal{ZT}^{vec}$. According to Proposition 5.10 and its proof, this can be done if the discrete logarithm problem, hence the representation problem, can be solved in $\mathbb{F}_{2^n}^*$. Proposition 5.12 even shows that an algorithm finding two paths in $\mathcal{ZT}^{proj}$ that end at the same vertex and start at a given random vertex is sufficient to find a cycle in $\mathcal{ZT}^{vec}$.

A cycle in $\mathcal{ZT}^{vec}$ induces cycles in both radial and angular coordinates. Propositions 5.10, 5.11 and 5.12 mean that solving the angular part of the representation problem is easy once the radial part can be solved to produce various points with the same radius.

The graphs $\mathcal{ZT}^{vec}$ and $\mathcal{ZT}^{proj}$ are quotient graphs of the graph $\mathcal{ZT}$, hence their expanding properties are at least as good. Indeed, let $f : \mathbb{F}_{2^n}^2 \setminus (0,0) \rightarrow \mathbb{R}$ be an eigenvector of the adjacency matrix of the vectorial version with eigenvalue μ: for any $v \in \mathbb{F}_{2^n}^2 \setminus (0,0)$ we have

$$f(v) = \mu(f(vA_0^{-1}) + f(vA_1^{-1})).$$

It follows that the vector $\hat{f}(M) : SL(2, \mathbb{F}_{2^n}) \rightarrow \mathbb{R} : \hat{f}(M) := f((1,0)M)$ is an eigenvector of the adjacency matrix of the matrix version with the same eigenvalue μ, because

$$\begin{aligned} \hat{f}(M) &= f((1,0)M) = \mu(f((1,0)MA_0^{-1}) + f((1,0)MA_1^{-1})) \\ &= \mu(\hat{f}(MA_0^{-1}) + \hat{f}(MA_1^{-1})). \end{aligned}$$

This shows that the largest eigenvalue of the vectorial version is at most as large as the corresponding eigenvalue of the matrix graph, hence the expansion of the vectorial version is at least as good as the expansion of the matrix version. By a similar argument, the expansion of the projective version is at least that of the vectorial version.

5.6 Cryptanalytic perspectives for ZT hash

We now discuss some cryptanalytic approaches that are promising for breaking the Zémor-Tillich hash function even if they have been unsuccessful so far. In particular, we introduce four problems that are all sufficient to solve in order to find collisions for the Zémor-Tillich hash function. We start with lifting attacks in Section 5.6.1 and we present other ideas in Section 5.6.2.

5.6.1 Lifting attacks

Lifting attacks have broken Zémor's first proposal and the LPS and Morgenstern hash functions, but they have been unsuccessful so far against the Zémor-Tillich hash function.

A lifting attack was developed by Tillich and Zémor [260] to solve the representation problem corresponding to the very first Cayley hash proposal by Zémor (Section 4.3.2). We recall that this scheme uses the matrix group $SL(2, \mathbb{F}_p)$ and the generators $S_0 = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ and $S_1 = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$. The idea of the attack is to lift the representation problem from $SL(2, \mathbb{F}_p)$ to $SL(2, \mathbb{Z})$.

The attack has two steps. First, a matrix $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in SL(2, \mathbb{Z})$ is found that reduces to the identity modulo p. The matrix M is then expressed as a product of S_0 and S_1 in $SL(2, \mathbb{Z})$, which can be done as follows: let $(x, y) = (a, b)$ if $a + b \geq c + d$, and $(x, y) = (c, d)$ otherwise. Apply the Euclidean algorithm to (x, y), that is, successively compute (q_1, r_1), (q_2, r_2), (q_3, r_3), *etc* such that $x = yq_1 + r_1$, $y = r_1 q_2 + r_2$, $r_1 = r_2 q_3 + r_3$, *etc.* These steps can also be written as $\left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & q_1 \\ 0 & 1 \end{smallmatrix}\right)\left(\begin{smallmatrix} r_1 \\ y \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)^{q_1}\left(\begin{smallmatrix} r_1 \\ y \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} r_1 \\ y \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)^{q_2}\left(\begin{smallmatrix} r_1 \\ r_2 \end{smallmatrix}\right)$, *etc* and so reveal the factorization of M.

The attack works well because all the nonnegative matrices of $SL(2, \mathbb{Z})$ can be factored as products of the matrices S_0 and S_1, and because it is possible to choose M such that the factorization is expected to be small [260].

A lifting attack was also found by Tillich and Zémor against the LPS hash function presented in Section 4.3.4. Their attack lifts the representation problem from $PSL(2, \mathbb{F}_p)$ to an appropriate subset Ω of $SL(2, \mathbb{Z}[i])$, with the property that any element of Ω has a unique factorization in the lifted generators [259]. Subsequently, we have extended Tillich and Zémor's algorithm to a preimage attack against LPS hash function and to the Morgenstern hash function [205]. The details of these attacks will be elaborated on in Chapter 6.

A lifting attack against the Zémor-Tillich hash function would lift the representation problem from $SL(2, \mathbb{F}_{2^n})$ to a subset Ω' of $SL(2, \mathbb{F}_2[X])$, such that any element (or a non-negligible proportion of the elements) of Ω' can be written as a product of $\widetilde{A}_0$ and $\widetilde{A}_1$, the lifts of A_0 and A_1 into $SL(2, \mathbb{F}_2[X])$. As we saw in Section 5.1.2, factorizing a matrix $M \in SL(2, \mathbb{F}_2[X])$ is easy when such a factorization exists, so the problem we are facing is to define Ω' together with an algorithm to find a lift of the identity in Ω.

Ideally, Ω' should be $\Omega := \langle\widetilde{A}_0, \widetilde{A}_1\rangle$, *i.e.* the set of matrices in $SL(2, \mathbb{F}_2[X])$ that is generated by $\widetilde{A}_0$ and $\widetilde{A}_1$. At least, $|\Omega \cap \langle\widetilde{A}_0, \widetilde{A}_1\rangle|/|\Omega|$ should not be too small and Ω should contain lifts of the identity. However, although the sets $\left\{\widetilde{A}^e, \forall e\right\}$ with $\widetilde{A} \in SL(2, \mathbb{F}_2[X])$ are very well understood (see Section 5.1.3), the set $\langle A_0, A_1\rangle$ cannot be characterized easily. We point out this problem as probably the most interesting research direction for the cryptanalysis of Zémor-Tillich hash function.

Problem 5.2 *Find a good characterization of* $\Omega = \langle\widetilde{A}_0, \widetilde{A}_1\rangle \in SL(2, \mathbb{F}_2[X])$.

5.6.2 Other ideas

One of the most successful approaches against Zémor-Tillich has been subgroup attacks. Subgroups specific to particular n values were first targeted by Steinwandt et al. and our goal in Section 5.4 was to exploit some subgroups of $SL(2, \mathbb{F}_{2^n})$ that exist for any n value. For well-chosen parameters, we do not expect any more serious threat to come from this side alone. However, subgroup attacks and in particular Proposition 5.10 could potentially be used to improve other attacks like Geiselmann's and the lifting attacks.

Let us first consider our variant of Geiselmann's attack. Let us remind that in this version, we were looking for exponents $\{e_i\}$ such that $A_0^{e_1}A_1^{e_2}A_0^{e_3}A_1^{e_4} = M$, where M is the hash of a random message and the other matrices are decomposed in Jordan form: $A_0 = S_0D_0S_0^{-1}$ where $D_0 = \begin{pmatrix} \lambda_0 & \\ & \lambda_0^{-1} \end{pmatrix}$ and $D_1 = \begin{pmatrix} \lambda_1 & \\ & \lambda_1^{-1} \end{pmatrix}$ are diagonal matrices (see Section 5.3.4). The resulting matrix equation $S_0D_0^{e_1}S_0^{-1}S_1D_1^{e_2}S_1^{-1}S_0D_0^{e_3}S_0^{-1}S_1D_1^{e_4}S_1^{-1} = M$ gives a system of four polynomial equations in the unknowns $x_1 = \lambda_0^{e_1}$, $x_2 = \lambda_1^{e_2}$, $x_3 = \lambda_0^{e_3}$ and $x_4 = \lambda_1^{e_4}$.

We may relax this system by requiring the matrix M to be *any* upper triangular matrix instead of the hash of a known message: according to Proposition 5.10, triangular matrices can be easily combined to produce the identity. The advantage of the new matrix equation is that it only gives one polynomial equation in the unknowns x_i (the equation constraining the lower left term of M to be 0), hence it has many other solutions including small ones. The problem of finding collisions for Zémor-Tillich now reduces to the following problem:

Problem 5.3 *Given l field elements $\lambda_i \in \mathbb{F}_{2^n}^*$ and a polynomial equation $f(x_1, ...x_l) = 0$ in the variables $x_i \in \mathbb{F}_{2^n}^*$, find $\{x_i = \lambda_i^{e_i}\}$ such that $f(x_1, ...x_l) = 0$ and $\sum |e_i|$ is* not too large.

Here is a somehow trivial way to solve this problem: Choose arbitrary "small" values (in the sense of the exponents being small) for all the x_i but two of them. Without loss of generality, we may assume that x_1 and x_2 remain to be fixed. Precompute $\lambda_2^{e_2}$ for $e_2 = 1, ..., 2^{n/2}$. For $e_1 = 1, ..., 2^{n/2}$ find x_2 such that $f(\lambda_1^{e_1}, x_2, x_3, ..., x_l) = 0$ and check whether this value appears in the list of precomputed values; repeat until finding one solution. This algorithm will succeed with a high probability but with time complexity, memory requirements and message lengths about $2^{n/2}$, that is worse than the algorithms of Section 5.4.

Problem 5.3 interestingly combines the additive and multiplicative laws of the field $\mathbb{F}_{2^n}$. Solving problems involving only the additive law of $\mathbb{F}_{2^n}$ is easy. In all generality, such a problem asks for finding $x_i \in \mathbb{F}_{2^n}$ such that $\sum a_i x_i = a_0$ for some $a_i \in \mathbb{F}_{2^n}$. As $\mathbb{F}_{2^n}$ is a vector space over $\mathbb{F}_2$, this problem amounts to solving a binary linear system which is very easy.

Solving problems involving only the multiplicative law of $\mathbb{F}_{2^n}$ is also easy for $n \leq 170$. In all generality, these problems are representation problems, which as shown in Proposition 5.9 can be solved if discrete logarithm problems can be solved. Although no efficient algorithm has been devised so far, discrete logarithm problems are well-understood and can be solved by subexponential algorithms that are practical for moderate n sizes.

Problem 5.3 mixes the additive and multiplicative laws of $\mathbb{F}_{2^n}$. It might help finding collisions for Zémor-Tillich as it seems reasonable that some particular instances have easy solutions. However, we expect Problem 5.3 to be harder in general than pure additive or pure multiplicative problems over $\mathbb{F}_{2^n}$, as we expect that neither the additive nor the multiplicative structure of $\mathbb{F}_{2^n}$ can be exploited to solve the polynomial equation.

The lifting attack could potentially also benefit from a mixed strategy: the attacker could lift random elements of a subgroup G' of $SL(2, \mathbb{F}_{2^n})$ instead of lifting the identity. If G' is the group of unimodular diagonal matrices, a group of unimodular triangular matrices, or the group of unimodular matrices with some given eigenvector, then the factorizations of random elements $g_i \in G'$ can be combined as in Proposition 5.10. This idea leads to the following relaxation of Problem 5.2.

Problem 5.4 *Find a good characterization of Ω', the subset of $\Omega = \langle \widetilde{A_0}, \widetilde{A_1} \rangle \in SL(2, \mathbb{F}_2[X])$ whose elements reduce to elements of G' modulo $P_n(X)$, where G' is the group of unimodular diagonal matrices, a group of unimodular tri-*

angular matrices, or the group of unimodular matrices with some given eigenvector.

We conclude our tour of attacks with an idea suggested in [205] to use the collision and preimage algorithms for the Morgenstern hash function (Section 4.3.5 and Chapter 6) to construct collisions and preimages for the Zémor-Tillich hash function when n is even. As $\langle s_0, s_1, s_2 \rangle = SL(2, \mathbb{F}_{2^n}) = \langle A_0, A_1 \rangle$ these algorithms could at first sight be of some help to find Zémor-Tillich collisions and preimages. At least, we can reduce the problem of finding preimages of any hash value to the problem of factoring three particular elements s_0, s_1, s_2, a potentially easier problem. In a first step, an adversary would construct one factorization of s_0, s_1 and s_2 as products of A_0 and A_1. In the second step, he would use the algorithms mentioned above for collisions and preimages and replace each occurrence of s_j in the solution by its corresponding factorization as a product of A_0 and A_1.

One idea to construct the factorization of s_0, s_1, s_2 as products of A_0, A_1 is to use N_{A_0}, N_{A_1} and N_I factorizations of A_0, A_1, I as products of s_0, s_1, s_2 and try to combine these factorizations. Let $\mathcal{M} := \{0,1,2\}^*/\sim$ where $\sim$ is the equivalence relation on $\{0,1,2\}^*$ defined by $m_1 \sim m_2$ if and only if m_2 can be constructed from m_1 by inserting or removing the sequences 00, 11 and 22 any times. The problem of finding collisions for Zémor-Tillich now reduces to the following problem.

Problem 5.5 *From N_{A_0}, N_{A_1} and N_I elements $a_j \in \mathcal{M}$, construct sequences m_1, m_2 and m_3 as concatenations of the a_j, such that m_1, m_2 and m_3 are in the same equivalence classes as respectively* 0, 1, *and* 2, *and have a size that is* not too large.

For small N_{A_0}, N_{A_1} and N_I, this problem is expected to have no solution. Indeed, let us define the *size* of an element $m \in \mathcal{M}$ as the smallest of the lengths of all the elements in the equivalence class of m. Then, for two elements $m_1, m_2 \in \mathcal{M}$ of size L, the size of $m_1||m_2$ is between 0 and $2L$ and is much more likely to be close to $2L$. The size of $m_1||m_2$ will be smaller than L only if more than one half of the last terms of m_1 coincide with the first terms of m_2. In mean, we need about $2^{L/2}$ messages m_2 to have one message inside able to reduce the size of m_1. As the sequences produced by the algorithms of [205] have length about $8n$, we believe that this approach cannot produce collisions nor preimages with reasonable sizes.

5.7 Is ZT hash secure?

Collision and preimage attacks are often considered successful (even if not practical) if they respectively beat the birthday paradox bound and the exhaustive search bound in time complexity. For Zémor-Tillich hash function these bounds are respectively $2^{3n/2}$ and 2^{3n}.

The practicability of a collision or preimage attack is usually measured by its computational time and its memory requirements. As discussed in Section 2.5.1, an attack with time complexity more than 2^{70} to 2^{80} or memory requirements larger than 2^{60} to 2^{70} cannot be executed today even by a large governmental agency. For Zémor-Tillich hash function, the existence of trivial but useless collisions suggests adding the message length as a quality criterium. Message lengths of 2^{40} to 2^{50} seem close to the practical limit, as they will appear only if large hard disks are hashed.

The trivial attack and Geiselmann's attacks (Section 5.3.4) can definitely not be considered as practical attacks, the message lengths being larger than or about 2^n. These attacks would become practical only for parameters n so small that even generic collision searches would also be practical. In order to perform the attacks of Sections 5.3.2 and 5.3.3, an adversary must choose the polynomial $P_n(X)$ himself, while the attacks of Section 5.3.5 only apply to weak parameters n.

To prevent all these attacks, the parameter n should be prime and the polynomial $P_n(x)$ should be chosen in a clearly honest way (for example, as the smallest irreducible polynomial of degree n or as the smallest irreducible polynomial larger than the polynomial whose coefficients are the truncated binary representation of π).

Our collision and preimage attacks of Section 5.4 are generic in the sense that they work for any parameter. With a temporal complexity close to $2^{n/2}$, they beat by far the ideal collision and preimage bounds $2^{3n/2}$ and 2^{3n}. The existence of these attacks suggests that the output of the Zémor-Tillich hash function should be of n bits rather than $3n$ bits.

For $n = 130$, the temporal complexity of our attacks is very close to practical bounds, hence we recommend to increase the parameter sizes by at least 30 bits. Taking $n = 251$ would give a very good security margin with respect to both computers and small attack improvements.

In Section 5.5, we have given two variants of the Zémor-Tillich hash function with output sizes respectively $2n$ and n, that are as secure as the original function for the parameters proposed.

Assuming the parameters are chosen as before, the only practical attacks against the Zémor-Tillich hash function are the generic malleability attack of Section 4.2.7 and the preimage attack for small messages that was presented in Section 5.3.1. These attacks do not contradict preimage or collision resistance of the function, but they discard its use in all applications where these properties are not sufficient, in particular all applications requiring a function behaving "like a random oracle".

In Section 5.6, we have described some tentative approaches that could eventually lead to an actual breaking of the function. Among all the ideas, techniques and approaches we have described in this chapter, we believe that lifting strategies are the most likely to succeed one day, possibly in combination with some subgroup approach. Besides, we also recommend to further investigate Problem 5.3 which nicely combines the additive and multiplicative laws of finite fields.

The Zémor-Tillich hash function somehow follows the block design of the Merkle-Damgard construction with message blocks of size 1 bit, the "compression function" being the matrix multiplication by A_0 or A_1 depending on the message bit. However, in Merkle-Damgard constructions, the compression function is usually supposed to have some cryptographic strength (in particular collision and preimage resistance) that the Zémor-Tillich compression function clearly does not have. As observed in Section 4.2.6, a direct consequence of this is the possibility of "meet-in-the-middle" attacks, such that computing preimages is not harder than finding collisions. Interestingly, our new subgroup collision attack of Section 5.4.2 could also be extended into a preimage attack with the same complexity.

The preimage and collision resistances of the Zémor-Tillich hash function reduce to the hardness of simply-stated mathematical problems, a very desired property of cryptographic algorithms. The problems here are however non-standard; they have been much less studied than discrete logarithm, integer factorization or elliptic curves discrete logarithm problems.

The (generalized) representation and the balance problems in $SL(2, \mathbb{F}_{2^n})$ have apparent weaknesses that were turned into partial attacks, and hash functions similar to the Zémor-Tillich hash have been completely broken. Hard and easy components of the representation problem for Zémor-Tillich have been clearly separated in Section 5.4.1. However, despite the numerous potential breaking approaches, none of them has really been damageable so far, and the approaches successful against other schemes could not be applied to the Zémor-Tillich hash function.

Today and 15 years after Zémor-Tillich publication, these problems remain essentially unbroken. We stress that they should be scrutinized again by the cryptographic community, in such a way that the function could become more trusted ... or completely broken.

Chapter 6

Cryptanalysis of LPS and Morgenstern hash functions

The LPS graph family was introduced into the expander hash construction by Charles et al. because of its "optimal" expanding properties [167, 68]: as discussed in Section 4.2.5, a large expansion guarantees a good output distribution of the hash function for relatively short messages. Subsequently, we proposed the Morgenstern hash function [206] with the aim of facilitating implementations and reducing the computational time.

In this Chapter, we show that neither the LPS nor the Morgenstern hash function are secure, in the sense that they are neither collision-resistant nor one-way. We start by describing the collision attack against LPS hash that was discovered by Tillich and Zémor [259]. We then extend this attack to a preimage attack against LPS hash, and we develop similar collision and preimage attacks against the Morgenstern hash function. We conclude with a few comments on how to repair LPS and Morgenstern hash functions and on the usefulness of our algorithms outside their original purposes.

This chapter is based on the work of Tillich and Zémor [259] and our work in collaboration with Kristin Lauter and Jean-Jacques Quisquater [204]. The results presented in [204] have been developed in two directions: first, further insight on the running times of our algorithm are given based on experimental results and second, the preimage attack against Morgenstern hash function has been fully developed. Examples of collision and preimage computations are given in Appendix E.

6.1 Tillich-Zémor collision attack against LPS hash

We recall from Section 4.3.4 that the LPS hash function is constructed from the non-bipartite LPS Ramanujan graph family [167, 68]. If p and l are primes, l is small and p is large, both p and l are equal to 1 mod 4 and l is a quadratic residue modulo p, these graphs are the Cayley graphs $\mathcal{C}_{G,S}$ defined by the group $G = PSL(2, \mathbb{F}_p)$ and the graph generators $S = \{s_j\}_{j=0,...,l}$, where

$$s_j = \begin{pmatrix} \alpha_j + \mathbf{i}\beta_j & \gamma_j + \mathbf{i}\delta_j \\ -\gamma_j + \mathbf{i}\delta_j & \alpha_j - \mathbf{i}\beta_j \end{pmatrix}, \qquad j = 0, ..., l;$$

$\mathbf{i}^2 = -1 \bmod p$ and $(\alpha_j, \beta_j, \gamma_j, \delta_j)$ are all the integer solutions of $\alpha^2 + \beta^2 + \gamma^2 + \delta^2 = l$, with $\alpha > 0$ and β, γ, δ even. Charles et al. recommend to use p of 1024 bits and $l = 5$, in which case the graph generators are (after reindexing such that $s_i^{-1} = s_{-i}$ for all i)

$$s_{\pm 1} = \begin{pmatrix} 1 \pm 2\mathbf{i} & 0 \\ 0 & 1 \mp 2\mathbf{i} \end{pmatrix}, \quad s_{\pm 2} = \begin{pmatrix} 1 & \pm 2 \\ \mp 2 & 1 \end{pmatrix} \quad s_{\pm 3} = \begin{pmatrix} 1 & \pm 2\mathbf{i} \\ \pm 2\mathbf{i} & 1 \end{pmatrix}.$$

In this section and the following one, we write i for the complex imaginary number satisfying $i^2 + 1 = 0$ and $\mathbf{i}$ for a solution to $\mathbf{i}^2 + 1 \equiv 0 \bmod p$. Moreover, we use small letters for elements of $\mathbb{Z}$ and corresponding capitalized letters for corresponding elements in $\mathbb{Z}_p$.

6.1.1 Outline of the attack

Tillich and Zémor's algorithm lifts the graph generators and the representation problem from $PSL(2, \mathbb{F}_p)$ to an appropriate subset Ω of $SL(2, \mathbb{Z}[i])$. This set Ω is defined by

$$\Omega = \left\{ \begin{pmatrix} a + bi & c + di \\ -c + di & a - bi \end{pmatrix} | (a, b, c, d) \in E_e \text{ for some integer } e > 0 \right\} \tag{6.1}$$

where E_e is the set of 4-tuples $(a, b, c, d) \in \mathbb{Z}^4$ such that

$$\begin{cases} a^2 + b^2 + c^2 + d^2 = l^e \\ a > 0, a \equiv 1 \bmod 2 \\ b \equiv c \equiv d \equiv 0 \bmod 2. \end{cases} \tag{6.2}$$

We call the first of these equations describing E_e the *norm equation*, as the left-hand side of this equation is the norm of the quaternion corresponding to the quadruplet (a, b, c, d) (see [167]). To each graph generator $s_j \in S$ is associated a lift $\widetilde{s}_j \in \widetilde{S} \subset \Omega$ of this graph generator defined by

$$\widetilde{s}_j = \begin{pmatrix} \alpha_j + i\beta_j & \gamma_j + i\delta_j \\ -\gamma_j + i\delta_j & \alpha_j - i\beta_j \end{pmatrix}, \qquad j = 0, ..., l.$$

The set Ω has two important properties: first, any element of Ω admits a unique factorization in terms of the lifts of the graph generators, and second, there exists a "reduction modulo p" multiplicative homomorphism φ from Ω to $PSL(2, \mathbb{F}_p)$ that allows translation of this factorization back to $PSL(2, \mathbb{F}_p)$.

Proposition 6.1 [167, 259] *Any matrix M in Ω can be expressed in a unique way as a product*

$$M = \pm l^r \widetilde{s}_{m_1} \widetilde{s}_{m_2} ... \widetilde{s}_{m_e}$$

where $\log_l(\det M) = e + 2r$, $\widetilde{s}_{m_j} \in \widetilde{S}$ for $j \in \{1, ..., e\}$ and $\widetilde{s}_{m_j} \widetilde{s}_{m_{j+1}} \neq lI$ for $j \in \{1, ..., e-1\}$.

Proposition 6.2 *The map $\varphi : \Omega \to PSL(2, \mathbb{F}_p)$*

$$\begin{pmatrix} a + bi & c + di \\ -c + di & a - bi \end{pmatrix} \to \begin{pmatrix} A + B\mathbf{i} & C + D\mathbf{i} \\ -C + D\mathbf{i} & A - B\mathbf{i} \end{pmatrix}$$

where $A = a \bmod p, B = b \bmod p, C = c \bmod p, D = d \bmod p$, is a multiplicative homomorphism. Therefore, any factorization $M = \widetilde{s}_{m_1} \widetilde{s}_{m_2} ... \widetilde{s}_{m_e} \in \Omega$ corresponds to a factorization $\varphi(M) = s_{m_1} s_{m_2} ... s_{m_e} \in PSL(2, \mathbb{F}_p)$.

In their exposition, Tillich and Zémor decompose their attack into three steps. The first step lifts the identity $I \in PSL(2, \mathbb{F}_p)$ to an element $\widetilde{I} \in \Omega$ such that $\varphi(\widetilde{I}) = \lambda I$ for some $\lambda \in \mathbb{F}_p^*$ with the additional condition that the lift is not a multiple of the identity in Ω. It amounts to finding integers a, b, c, d and λ satisfying the following conditions:

$$\left\{ \begin{array}{lcl} (a, b, c, d) & \in & E_e \\ (a, b, c, d) & \not\equiv & (0, 0, 0, 0) \bmod l \\ (a, b, c, d) & \equiv & \lambda(1, 0, 0, 0) \bmod p. \end{array} \right. \tag{6.3}$$

Putting every congruence condition into the norm equation leads to a diophantine equation whose resolution by Tillich and Zémor will be presented in Section 6.1.2.

The second step of the attack is to factorize the lifted element $\widetilde{I}$ of Ω into products of lifted generators $\widetilde{s}_j, j = 1, ..., l + 1$. By Proposition 6.1, we know that this factorization is unique and has size e, so let us write it

$\widetilde{I} = \widetilde{s}_{j_1}\widetilde{s}_{j_2}...\widetilde{s}_{j_e}$. Multiplying on the right by a lifted generator $\widetilde{s} \in \widetilde{S}$ gives a matrix that is divisible by l if and only if $\widetilde{s} = (\widetilde{s}_{j_e})^{-1}$ so by trying each of the graph generators we get the last factor, and we then proceed recursively.

The final step translates the factorization of I' in Ω into a factorization of the identity in $PSL(2, \mathbb{F}_p)$. By using the homomorphism φ defined in Proposition 6.2, this last step is trivial.

6.1.2 Solving the equation

We now sketch the algorithm used by Tillich and Zémor to solve System 6.3. The exponent e is arbitrarily put to be even, that is $e = 2k$, where k is the smallest integer such that $l^k - 4p^2 > 0$. Combining the equations of Systems 6.2 and 6.3, we may write $b = 2xp$, $c = 2yp$ and $d = 2zp$ for some integers x, y, z; the only constraints not taken into account so far are the norm equation, the constraint $a > 0, a \equiv 1 \bmod 2$ and the condition that the lift is not a multiple of identity (which will not be checked until the end). The norm equation can be written as

$$a^2 + 4x^2p^2 + 4y^2p^2 + 4z^2p^2 = l^{2k} \tag{6.4}$$

hence

$$(l^k - a)(l^k + a) = 4p^2(x^2 + y^2 + z^2).$$

Fixing $a = l^k - 2mp^2$ which is odd and positive by definition of k, the norm equation may be "simplified by $4p^2$" to lead to the equation

$$x^2 + y^2 + z^2 = m(l^k - mp^2).$$

Using Legendre's theorem [140], Tillich and Zémor have shown that this equation has solutions either when m is equal to 0 or 1. A solution can be found by assigning random values to x until the resulting equation

$$y^2 + z^2 = n := m(l^k - mp^2) - x^2 \tag{6.5}$$

has solutions for integers y, z, which can be verified by Fermat's theorem [140]: $y^2 + z^2 = n$ has solutions if and only if all the prime factors of n congruent to 3 modulo 4 occur with an even exponent in the factorization of n. In particular, and this is easier to check as it does not require any factorization step, the equation has solutions if $n = 2^s p'$ for some prime p congruent to 1 modulo 4.

Solving Equation (6.5) can be done very efficiently if solutions exist. When n is prime, a solution can be found with the continuous fraction algorithm which is essentially the Euclidean algorithm (we refer to [259] for more details). The composite case reduces to the prime case as if $y_1^2 + z_1^2 = n_1$ and $y_2^2 + z_2^2 = n_2$ then $(y_3, z_3) = (y_1y_2 - z_1z_2, z_1y_2 + y_1z_2)$ satisfies $y_3^2 + z_3^2 = n_1n_2$.

6.1.3 Runtime of the algorithm

Tillich and Zémor estimate that their algorithm runs in time $O(\log p)$. They argue that the number of x's values that have to be tested in order to find a proper n value (a value which is the sum of two squares) is about $O(\log p)$ and the continuous fraction expansion also requires time $O(\log p)$. Their algorithm is therefore very efficient and it is definitely practical when p has 1024 bits as prescribed by [68].

6.2 Preimages for the LPS hash function

Suppose now that we are given a matrix $M = \left(\begin{smallmatrix} M_1 & M_2 \\ M_3 & M_4 \end{smallmatrix}\right) \in PSL(2, \mathbb{F}_p)$ which has square determinant, and we are asked to find a preimage, that is a factorization of it with the graph generators. By solving two linear equations in $\mathbb{F}_p$ we can write M in the form

$$M = \begin{pmatrix} A + B\mathbf{i} & C + D\mathbf{i} \\ -C + D\mathbf{i} & A - B\mathbf{i} \end{pmatrix}.$$

6.2.1 Outline of the attack

Our algorithm follows along the lines of Zémor and Tillich's. We first lift the problem from $PSL(2, \mathbb{F}_p)$ to the set Ω defined above, then we factorize in Ω and we finally come back to $PSL(2, \mathbb{F}_p)$. The only difference is in the first step. Lifting the representation problem now amounts to finding integers a, b, c, d and λ satisfying the following conditions:

$$\begin{cases} (a,b,c,d) & \in & E_e \\ (a,b,c,d) & & \text{not divisible by } l \\ (a,b,c,d) & \equiv & \lambda(A,B,C,D) \bmod p. \end{cases}$$

We write $a = A\lambda + wp$, $b = B\lambda + xp$, $c = C\lambda + yp$ and $d = D\lambda + zp$ with $w, x, y, z \in \mathbb{Z}$. For convenience we choose e even, that is $e = 2k$ for k an integer. The norm equation becomes

$$(A\lambda + wp)^2 + (B\lambda + xp)^2 + (C\lambda + yp)^2 + (D\lambda + zp)^2 = l^{2k}. \tag{6.6}$$

At first sight, this equation seems much harder to solve than Equation (6.4). In the case $B = C = D = 0$, the norm equation is Equation (6.4) which was solved by Zémor and Tillich as recalled in Section 6.1.2. Their algorithm fixes the value of $a = A\lambda + wp$ in an appropriate way that allows "simplifying the equation by p^2". In Equation (6.6), we cannot fix a independently and then divide by p^2 because of the term $2p(wA + xB + yC + zD)\lambda$.

Since we do not simplify by p^2, the coefficients of degree-2 terms are huge (at least p), and the resulting equation is at first sight very hard to solve.

We overcome this difficulty with a new idea. In the remainder of this section, we will solve the preimage problem for diagonal matrices with A and/or B non-zero, and then we will write any matrix as a product of four diagonal matrices and up to four graph generators. Altogether this leads to an efficient probabilistic algorithm that finds preimages for the LPS hash function.

6.2.2 Preimages for diagonal matrices

Now we show how to find a factorization of a matrix

$$M = \begin{pmatrix} A + B\mathbf{i} & \\ & A - B\mathbf{i} \end{pmatrix}$$

such that $A^2 + B^2$ is a square modulo p. Write $y = 2y'$ and $z = 2z'$ where y', z' are integers. We need to find integer solutions to

$$\begin{cases} (A\lambda + wp)^2 + (B\lambda + xp)^2 + 4p^2(y'^2 + z'^2) = l^{2k} \\ A\lambda + wp \equiv 1 \bmod 2 \\ B\lambda + xp \equiv 0 \bmod 2 \end{cases}$$

Fix $k = \lceil \log_l(8p^2) \rceil$. As $A^2 + B^2$ is a square, there are exactly two values for λ in $\{0, 1, ...p - 1\}$ satisfying the norm equation modulo p:

$$(A^2 + B^2)\lambda^2 = l^{2k} \bmod p.$$

We choose either of them, and we let $m := (l^{2k} - (A^2 + B^2)\lambda^2)/p$. Our strategy is to pick random solutions of

$$\begin{cases} l^{2k} - (A\lambda + wp)^2 - (B\lambda + xp)^2 \equiv 0 \bmod p^2 \\ A\lambda + wp \equiv 1 \bmod 2 \\ B\lambda + xp \equiv 0 \bmod 2 \end{cases}$$

until the equation

$$4(y'^2 + z'^2) = n$$

has solutions, where

$$n := \left(l^{2k} - (A\lambda + wp)^2 - (B\lambda + xp)^2\right)/p^2.$$

A random solution to the congruence system is computed as follows: until you get x with the correct parity, pick a random $w \in \{0, 1, ...p - 1\}$ with the right parity and compute $x = \frac{m}{2\lambda B} - \frac{A}{B}w \bmod p$. By the way k, x and w are chosen we are sure that $n > 0$ so the equation $4(y'^2 + z'^2) = n$ has solution if and only if 4 divides n and all prime factors of n congruent to 3 modulo 4 appear an even number of times in the factorization of n. To avoid the factorization of n in the algorithm, we will actually strengthen this condition to n being equal to 4 times a prime congruent to 1 modulo 4. When it has solutions, the equation $4(y'^2 + z'^2) = n$ is easily solved with the Euclidean algorithm, as recalled in [259]. After lifting the problem to $SL(2, \mathbb{Z}[i])$ the second and third steps of the algorithm are the same as in Tillich-Zémor algorithm. So we are done with the factorization of diagonal matrices.

6.2.3 Reduction to the diagonal case

Now we show how to decompose any matrix $M \in PSL(2, \mathbb{F}_p)$ into a product of diagonal matrices and graph generators. We may additionally assume that all the entries of M are nonzero: if they are not, just multiply M by ss^{-1} for some adequate s in S, and consider the factorization of $s^{-1}M$. We will show how to find (λ, α, ω, β_1, β_2) with the last four being squares, such that

$$\begin{pmatrix} M_1 & M_2 \\ M_3 & M_4 \end{pmatrix} = \lambda \begin{pmatrix} 1 & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} = \lambda \begin{pmatrix} f_1 & \omega f_2 \\ \alpha f_3 & \alpha\omega f_4 \end{pmatrix} \tag{6.7}$$

and

$$\begin{aligned} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} &= \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 - 4\beta_1 - 4\beta_2 - 4\beta_1\beta_2 & 2 - 8\beta_1 + 2\beta_2 + 2\beta_1\beta_2 \\ -2 - 2\beta_1 + 8\beta_2 - 2\beta_1\beta_2 & -4 - 4\beta_1 - 4\beta_2 + \beta_1\beta_2 \end{pmatrix}. \end{aligned}$$

Lemma 6.1 *Matrix equation (6.7) is equivalent to the following system:*

$$\begin{cases} M_2M_3f_1f_4 - M_1M_4f_2f_3 & = 0 \\ \alpha M_1f_3 - M_3f_1 & = 0 \\ \omega M_3f_4 - M_4f_3 & = 0 \\ \lambda f_1 - M_1 & = 0 \end{cases} \tag{6.8}$$

PROOF: ($\Rightarrow$) Fourth equation is entry (1,1) of the matrix equation. Third equation is entry (2,1) times M_1 minus entry (1,1) times M_3. Second equation is entry (1,2) times M_1 minus entry (1,1) times M_2. First equation is entry (1,1) times entry (2,2) times M_2M_3 minus entry (1,2) times entry (2,1) times M_1M_4.

($\Leftarrow$) Last equation is $M_1 = \lambda f_1$ that is entry (1,1). We have $M_2 = \frac{M_1 M_4 f_2 f_3}{M_3 f_1 f_4}$ by first equation so $M_2 = f_2 \frac{M_4 f_3}{M_3 f_4} \frac{M_1}{f_1} = f_2 \omega \lambda$ by third and fourth equation, that is entry (1,2). We have $M_3 = \frac{\alpha M_1 f_3}{f_1} = \alpha \lambda f_3$ by second then fourth equation, that is entry (2,1). We have $M_4 = \omega M_3 \frac{f_4}{f_3}$ by third equation, so using the already proved entry (2,1) we have $M_4 = \omega \alpha \lambda f_3 \frac{f_4}{f_3} = \omega f_4 \alpha \lambda$ that is entry (2,2). □

In the system of equations (6.8), the first equation only involves β_1 and β_2 while the other equations are linear once β_1 and β_2 are fixed. So we can concentrate on solving the first equation, which is quadratic in both β_1 and β_2:

$$\begin{aligned} M_2 M_3 f_1 f_4 \quad - \quad & M_1 M_4 f_2 f_3 = 4(M_2 M_3 - M_1 M_4)(-\beta_1^2 + 3\beta_1 + 4)\beta_2^2 \\ & + \left(M_2 M_3 (12\beta_1^2 + 49\beta_1 + 12) + M_1 M_4 (-12\beta_1^2 + 76\beta 1 - 12)\right)\beta_2 \\ & +4(M_2 M_3 - M_1 M_4)\left(4\beta_1^2 + 3\beta_1 - 1\right). \end{aligned}$$

Our algorithm then proceeds as follows:

1. Pick a random β_1 which is a square.
2. Compute the discriminant of the quadratic equation in β_2, β_1. If it is not a square, go back to 1.
3. Solve the quadratic equation. If none of the roots is a square, go back to 1. Else, assign a quadratic root to β_2.
4. Compute f_1, f_2, f_3, f_4.
5. Solve $\alpha M_1 f_3 - M_3 f_1 = 0$ to get α. If α is not a square, go back to 1.
6. Solve $\omega M_3 f_4 - M_4 f_3 = 0$ to get ω. If ω is not a square, go back to 1.

This concludes the exposition of our algorithm. Examples are given in Appendix E.

6.2.4 Runtime analysis

First consider the algorithm for diagonal matrices. Assuming n behaves "as a random number" then according to the prime number theorem we will need $\mathcal{O}(\log n) = \mathcal{O}(\log p)$ trials before getting one n of the correct form. For each trial, the most expensive computation is a primality test, which can be done in polynomial time (in our implementation, we actually use the probabilistic function *mpz_probab_prime_p* of GNU MP). So the algorithm for diagonal matrices is probabilistic polynomial time. In the reduction algorithm, the probability for a random number to be a square modulo p being one half, we estimate that a solution (λ, α, ω, β_1, β_2) with the last four being squares can be found in about 2^4 trials. Consequently, the whole algorithm is probabilistic polynomial time.

We have implemented this algorithm in C using GNU MP. With this implementation, finding preimages for 1024-bit parameters on an Intel Pentium R 4CPU processor 3.20GHz requires between 10 and 40 seconds. Additional timings for other n values are given in Figure 6.1.

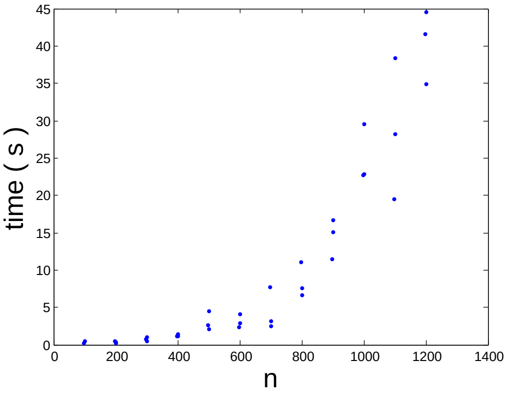

Figure 6.1: Computation times of our preimage algorithm for LPS hash function for various parameters n

6.3 Collisions for the Morgenstern hash function

We now adapt the algorithms of Section 6.1 and 6.2 to the Morgenstern hash function. We recall from Section 4.3.5 that this hash function is a Cayley hash constructed from the Morgenstern Ramanujan graph family for binary fields [190, 206]. Let q be a power of 2 and $f(X) = X^2 + X + \epsilon$ irreducible in $\mathbb{F}_q[x]$. Let $P_n(x) \in \mathbb{F}_q[x]$ be irreducible of even degree $n = 2d$ and let $\mathbb{F}_{q^n}$ be represented by $\mathbb{F}_q[x]/(P_n(x))$. Morgenstern graphs are the Cayley graph $\mathcal{C}_{G,S}$ defined by the group $G = PSL(2, \mathbb{F}_{2^n})$ and the graph generators $S = \{s_j\}_{j=0,...q}$ where

$$s_j = \begin{pmatrix} 1 & \gamma_j + \delta_j \mathbf{i} \\ X(\gamma_j + \delta_j \mathbf{i} + \delta_j) & 1 \end{pmatrix}, \qquad j = 0, ..., q;$$

and $\gamma_j, \delta_j \in \mathbb{F}_q$ are all the $q+1$ solutions in $\mathbb{F}_q$ for $\gamma_j^2 + \gamma_j\delta_j + \delta_j^2\epsilon = 1$.

In this section and the following one, we focus on the case $q = 2$ but the attacks generalize easily to other powers of 2. When $q = 2$, $f(X) = X^2+X+1$ and the 3 graph generators are

$$s_0 = \begin{pmatrix} 1 & 1+\mathbf{i} \\ X\mathbf{i} & 1 \end{pmatrix}, \quad s_1 = \begin{pmatrix} 1 & 1 \\ X & 1 \end{pmatrix}, \quad s_2 = \begin{pmatrix} 1 & \mathbf{i} \\ X(1+\mathbf{i}) & 1 \end{pmatrix}.$$

Let $\mathbb{A} := \mathbb{F}_2[X,Y]/(Y^2+Y+1)$. In this section and the following, we denote by i a root of $i^2+i+1=0$ in $\mathbb{A}$ and by $\mathbf{i}$ a root of the same equation in $\mathbb{F}_{2^n}$ (which has a solution by Lemma 6.2 below). Moreover, we use small letters for elements of $\mathbb{F}_2[X]$ and the corresponding capitalized letters for the corresponding "modulo $P_n(X)$" elements in the field $\mathbb{F}_{2^n}$. Finally, we write p for $P_n(X)$ both to lighten notations and to emphasize the similarity with the previous sections.

6.3.1 Outline of the attack

Our algorithm lifts the representation problem from $SL(2, \mathbb{F}_{2^n})$ to a subset Ω of $SL(2, \mathbb{A})$. The relevant set is

$$\Omega = \left\{ \begin{pmatrix} a+bi & c+di \\ X(c+di+d) & a+bi+b \end{pmatrix} |(a,b,c,d) \in E_e \text{ for some integer } e > 0 \right\}$$

where E_e is the set of 4-tuples $(a,b,c,d) \in \mathbb{F}_2[X]$ such that

$$\begin{cases} (a^2+b^2+ab) + (c^2+d^2+cd)X = (1+X)^e \\ a \equiv 1 \bmod X \\ b \equiv 0 \bmod X. \end{cases}$$

We again call the first of these equations the *norm equation.* By [190], corollary 5.4 and 5.7, if we restrict E_e to tuples (a,b,c,d) not divisible by $(1+X)$, the elements of Ω have a unique factorization in terms of the lifts of the graph generators:

$$\widetilde{s_0} = \begin{pmatrix} 1 & 1+i \\ Xi & 1 \end{pmatrix}, \quad \widetilde{s_1} = \begin{pmatrix} 1 & 1 \\ X & 1 \end{pmatrix}, \quad \widetilde{s_2} = \begin{pmatrix} 1 & i \\ X(1+i) & 1 \end{pmatrix}.$$

Moreover, the "reduction modulo p" $(a,b,c,d) \to (A,B,C,D) = (a,b,c,d) \bmod p$ gives a homomorphism from Ω to $SL(2, \mathbb{F}_{2^n})$:

$$\begin{pmatrix} a+bi & c+di \\ X(c+di+d) & a+bi+b \end{pmatrix} \to \begin{pmatrix} A+B\mathbf{i} & C+D\mathbf{i} \\ X(C+D\mathbf{i}+D) & A+B\mathbf{i}+B \end{pmatrix}.$$

From this it is now clear how the second and third steps of Tillich and Zémor's algorithm will work for Morgenstern hashes, so we now give details for the first step. This amounts to lifting the representation problem, that is finding $a, b, c, d, \lambda \in \mathbb{F}_{2^n}$ satisfying the following conditions:

$$\begin{cases} (a,b,c,d) & \in & E_e \\ (a,b,c,d) & \not\equiv & (0,0,0,0) \bmod X+1 \\ (a,b,c,d) & \equiv & \lambda(1,0,0,0) \bmod p. \end{cases}$$

Write $b = Xpb'$, $c = pc'$, $d = pd'$ for $b', c', d' \in \mathbb{F}_2[X]$ and arbitrarily choose $e = 2k$ and $a = (1+X)^k + Xpm$, with $k \in \mathbb{Z}$ and $m \in \mathbb{F}_2[X]$ still to be determined. Note that such an a satisfies $a \equiv 1 \bmod X$. The norm equation becomes

$$X^2p^2m^2 + X^2p^2b'^2 + Xpb'\left((1+X)^k + mXp\right) + Xp^2(c'^2 + d'^2 + c'd') = 0.$$

Simplifying by Xp, we get

$$Xpm^2 + Xpb'^2 + b'\left((1+X)^k + Xpm\right) + p(c'^2 + d'^2 + c'd') = 0.$$

Reducing this equation modulo p we get $b'\left((1+X)^k + Xpm\right) \equiv 0$ which implies that $b' = pb''$ for some $b'' \in \mathbb{F}_2[X]$. The norm equation becomes

$$Xpm^2 + Xp^3b''^2 + pb''\left((1+X)^k + Xpm\right) + p(c'^2 + d'^2 + c'd') = 0.$$

Simplifying again by p, we get

$$c'^2 + d'^2 + c'd' = n(b'', m, k) := Xm^2 + Xp^2b''^2 + b''(1+X)^k + b''Xpm.$$

Our approach for step 1 will be to generate random m and b'' (with $X+1 \nmid b''$) until the equation $c'^2 + d'^2 + c'd' = n(b'', m, k)$ has solutions, then to solve this equation for c', d'. As will be clear in Section 6.3.2, the equation has a solution if and only if *all the irreducible factors of n are of even degree.* So in particular

- We will choose $b'' = b^{(3)}X + 1$ for some $b^{(3)} \in \mathbb{F}_2[X]$ to avoid an X factor.
- As the term $Xp^2b''^2$ is of odd degree, we will make another term of higher even degree, with the following strategy:
  - Choose b'' and m randomly of degree equal to or less than R.
  - Choose $k = 2\deg(p) + \deg(b'') + 2 + (\deg(b'') + \epsilon)$ where $\epsilon = 0$ if $\deg(b'')$ is even and $\epsilon = 1$ if $\deg(b'')$ is odd.

If R is large enough we get an n with the desired property after sufficiently many random trials on b'' and m. In our implementation, we chose $R = 10$ which is more than enough for 1024-bit parameters. It remains to show how to solve the equation $c'^2 + d'^2 + c'd' = n$ and to explain the condition on the degrees of the irreducible factors of n. We begin with the solution of the equation.

6.3.2 Solving $c^2 + d^2 + cd = n$

We first remark that it is enough to have an algorithm solving it when n is irreducible. Indeed, if $c_1^2 + d_1^2 + c_1 d_1 = n_1$ and $c_2^2 + d_2^2 + c_2 d_2 = n_2$ then $(c_3, d_3) = (c_1 c_2 + d_1 d_2, c_1 d_2 + c_2 d_1 + d_1 d_2)$ satisfies $c_3^2 + d_3^2 + c_3 d_3 = n_1 n_2$. So suppose n is irreducible of even degree.

We describe a continued fraction algorithm for polynomials over $\mathbb{F}_2$ and then we use it to solve the equation. For a fraction $\xi = \frac{P}{Q}$ where P and Q are polynomials, let $P = a_0 Q + r_0$ where $\deg r_0 < \deg Q$. Let $Q = a_1 r_0 + r_1$ with $\deg r_1 < \deg r_0$, then recursively for $i = 2, \ldots$, define $r_{i-2} = a_i r_{i-1} + r_i$ with $\deg r_i < \deg r_{i-1}$. (This is the Euclidean algorithm applied to the ring $\mathbb{F}_2[X]$.) Define $p_0 = a_0$, $q_0 = 1$, $p_1 = a_0 a_1 + 1$, $q_1 = a_1$, and then recursively $p_i = a_i p_{i-1} + p_{i-2}$ and $q_i = a_i q_{i-1} + q_{i-2}$. (The fraction p_i/q_i is the i^{th} truncated continued fraction of P/Q.) We see recursively that $q_i p_{i-1} + q_{i-1} p_i = 1$, so $\frac{p_i}{q_i} + \frac{p_{i-1}}{q_{i-1}} = \frac{1}{q_{i-1} q_i}$ and

$$\frac{P}{Q} = a_0 + \sum_{i=0}^{n} \frac{1}{q_{i+1} q_i}$$

where n is the first i such that $p_i/q_i = P/Q$. Consider the valuation v on rational fractions defined as follows: $v\left(\frac{a}{b}\right) = \deg b - \deg a$ if $a, b \neq 0$, and $v\left(\frac{a}{b}\right) = \infty$ if $a = 0, b \neq 0$. Note that $v(q_{i+1}) \leq v(q_i)$, $v(p_{i+1}) \leq v(p_i)$, and that

$$v\left(\frac{P}{Q} + a_0 + \sum_{i=0}^{n'-1} \frac{1}{q_{i+1} q_i}\right) = v\left(\sum_{i=n'}^{n} \frac{1}{q_{i+1} q_i}\right) \geq -v(q_{n'+1}) - v(q_{n'}).$$

As n has even degree, we can compute $\alpha \in \mathbb{F}_2[X]$ such that $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$ (see Section 6.3.3). We apply a continued fraction expansion to $\xi = \frac{\alpha}{n}$ and let p_i/q_i be the successive approximations. Let j be such that

$$v(q_j) \geq \frac{v(n)}{2} \geq v(q_{j+1}).$$

We have

$$q_j^2 + (q_j \alpha + p_j n)^2 + q_j (q_j \alpha + p_j n) \equiv q_j^2 + q_j^2 \alpha^2 + q_j^2 \alpha \equiv 0 \bmod n.$$

On the other hand, as

$$\begin{aligned}
-\deg(q_j\alpha + p_j n) &= v(n) + v(q_j) + v\left(\xi + \frac{p_j}{q_j}\right) \\
&\geq v(n) + v(q_j) - v(q_j) - v(q_{j+1}) \geq v(n)/2 = -\deg(n)/2
\end{aligned}$$

we have

$$v\left(q_j^2 + (q_j\alpha + p_i n)^2 + q_i(q_j\alpha + p_j n)\right) \geq 2\max\left(\deg(q_j), \deg(q_j\alpha + p_j n)\right) \geq -\deg n.$$

Consequently,

$$q_j^2 + (q_j\alpha + p_j n)^2 + q_j(q_j\alpha + p_j n) = n$$

and $(c,d) = (q_j, q_j\alpha + p_j n)$ is a solution to $c^2 + d^2 + cd = n$.

6.3.3 Solutions to $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$

We now show that the equation $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$ has solutions over $\mathbb{F}_2[X]$ for n irreducible if and only if the degree of n is even, and that in that case a solution can be efficiently computed.

As the map $\alpha \to \alpha^2 + \alpha$ is a linear application acting on the vector space $\mathbb{F}_{2^n}/\mathbb{F}_2$, solutions to this equation, if there are any, are found easily by writing down then solving a linear system of equations. We conclude the exposition of our algorithm by showing the following lemma.

Lemma 6.2 *For n irreducible, $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$ has solutions if and only if $d := \deg(n)$ is even.*

PROOF:[1] ($\Rightarrow$) Suppose α satisfies $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$. Then $1 = \alpha + \alpha^2$. Squaring each side we get $1 = \alpha^2 + \alpha^{2^2}$, then squaring again and again we get $1 = \alpha^{2^2} + \alpha^{2^3}$,... until $1 = \alpha^{2^d} + \alpha^{2^{d-1}} = \alpha + \alpha^{2^{d-1}}$. Summing up these equations we get $d = 0$, so d must be even.

($\Leftarrow$) Now suppose d is even. Let β be a generator of $\mathbb{F}_{2^d}^*$ and let $\alpha = \beta^{\frac{2^d-1}{3}}$. Then $\alpha^3 = 1$ and $\alpha \neq 1$ so $\alpha^2 + \alpha + 1 = 0$. □

This concludes the description of our collision algorithm for Morgenstern hash function. Intermediary results of the algorithm on toy parameters and on parameters of 1024 bits are given in Appendix E.

[1] The lemma is a particular case of a standard result in finite field theory, stating that $\mathbb{F}_{p_1^d} \subset \mathbb{F}_{p_2^d}$ if and only if $d_1|d_2$.

6.3.4 Runtime analysis

We give some estimates for the complexity of our algorithm. Assuming the polynomial n generated from random (b'', m) behaves like random polynomials of degree k, the number of its irreducible factors is asymptotically $K = \mathcal{O}(\log \deg n)$ [106]. For n of degree even, we can reasonably approximate the probability that all its factors are of even degree by $(1/2)^K$, hence we will need $2^K = \mathcal{O}(\deg n) = \mathcal{O}(\deg p)$ random trials. The factorization of n can be done in $\mathcal{O}(\log^{2+\epsilon} n)$ [245] and the continued fraction algorithm is of complexity $\mathcal{O}(\deg n)$, so the global complexity of our algorithm is probabilistic polynomial time in $\deg p$.

We have implemented this algorithm in C++ using NTL. With this implementation, finding preimages for 1024-bit parameters requires between 10 and 25 seconds on an Intel Pentium R 4CPU processor 3.20GHz. Additional timings for other n values are given in Figure 6.2.

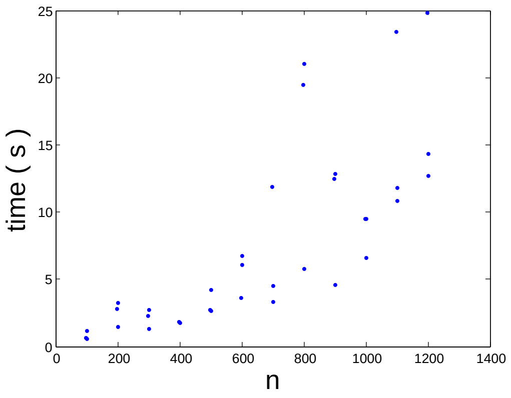

Figure 6.2: Computation times of our collision algorithm for Morgenstern hash function for various parameters n

6.4 Preimages for the Morgenstern hash function

We now combine the ideas of the previous sections to produce a preimage algorithm against the Morgenstern hash function. We focus again on $q = 2$ but extensions to other powers of 2 are easy.

Suppose we are given a matrix $M = \left(\begin{smallmatrix} M_1 & M_2 \\ M_3 & M_4 \end{smallmatrix}\right) \in PSL(2, \mathbb{F}_{2^n})$ and we are asked to find a preimage, that is a factorization of it with the graph generators. By solving two linear equations in $\mathbb{F}_{2^n}$ we can write it in the form

$$M = \begin{pmatrix} A + B\mathbf{i} & C + D\mathbf{i} \\ x(C + D\mathbf{i} + D) & A + B\mathbf{i} + B \end{pmatrix}.$$

As most of the algorithm is a straightforward merge between the algorithms of Sections 6.2 and 6.3, we have not implemented it but we provide some runtime analysis. Like in Section 6.2, we first show how to produce preimages of diagonal matrices and then reduce the general case to the diagonal case.

6.4.1 Preimages of diagonal matrices

With the lifting strategy, finding a preimage to a matrix

$$M = \begin{pmatrix} A + B\mathbf{i} & \\ & A + B\mathbf{i} + B \end{pmatrix}$$

amounts to finding $\lambda, a, b, c, d \in \mathbb{F}_2[X]$ such that

$$\begin{cases} (a, b, c, d) & \in & E_e \\ (a, b, c, d) & \not\equiv & (0, 0, 0, 0) \bmod X + 1 \\ (a, b, c, d) & \equiv & \lambda(A, B, 0, 0) \bmod p. \end{cases}$$

Writing $a = A\lambda + wp$, $b = B\lambda + xp$, $c = yp$ and $d = zp$, the norm equation becomes

$$(A\lambda + wp)^2 + (B\lambda + xp)^2 + (A\lambda + wp)(B\lambda + xp) + p^2(y^2 + z^2 + yz) = (1 + X)^e$$

or

$$(A^2+B^2+AB)\lambda^2+(Aw+Bx)\lambda p+(w^2+x^2+wx)p^2+(y^2+z^2+yz)p^2 = (1+X)^e.$$

Fix $e = 2k$ with $k = 2\deg(p) + 1$. The value of λ is fixed by reducing the equation modulo p: λ is the square root modulo p of $(1+X)^e/(A^2+B^2+AB)$. We then pick random values of (w, x) satisfying the equation modulo p^2 and such that $A\lambda + wp \equiv 1 \bmod X$ and $B\lambda + xp \equiv 0 \bmod X$, until all irreducible factors of

$$n := [(A^2 + B^2 + AB)\lambda^2 + (Aw + Bx)\lambda p + (w^2 + x^2 + wx)p^2 + (1 + X)^e]/p^2$$

have even degree. (By our choice of k, n has even degree equal to $4\deg p+2$.) After $O(\deg n) = O(\deg p)$ random trials, we are likely to get n of the correct form (see Section 6.3.4) after which we can solve the equation $y^2+z^2+yz = n$ as in Section 6.3.

6.4.2 Reduction to the diagonal case

Now we show how to decompose any matrix $M \in PSL(2, \mathbb{F}_{2^n})$ into a product of diagonal matrices and graph generators. We may additionally assume that all the entries of M are nonzero: if they are not, just multiply M by ss^{-1} for some adequate s in S, and consider the factorization of $s^{-1}M$. We will show how to find (λ, α, ω, β_1, β_2) such that

$$\begin{pmatrix} M_1 & M_2 \\ M_3 & M_4 \end{pmatrix} = \lambda \begin{pmatrix} 1 & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} = \lambda \begin{pmatrix} f_1 & \omega f_2 \\ \alpha f_3 & \alpha\omega f_4 \end{pmatrix} \tag{6.9}$$

and

$$\begin{aligned} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ X & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ X & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ X & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 + (\beta_1 + \beta_2 + \beta_1\beta_2)X & 1 + \beta_1 X + \beta_2 X + \beta_1\beta_2 \\ X + (\beta_1 + \beta_2 + \beta_1\beta_2)X & \beta_1\beta_2 + X(1 + \beta_1 + \beta_2) \end{pmatrix}. \end{aligned}$$

It is straightforwardly checked that Lemma 6.1 generalizes to Equation (6.9) hence we are left with solving the equation $M_2M_3f_1f_4 + M_1M_4f_2f_3 = 0$ which is quadratic in β_1 and β_2, after which we get λ, α and ω by solving three linear equations. We have

$$\begin{aligned} M_2M_3f_1f_4 \quad + \quad & M_1M_4f_2f_3 = X(M_2M_3 + M_1M_4)(\beta_1 + 1)(\beta_1 + X)\beta_2^2 \\ & + \left[M_2M_3(X^2\beta_1^2 + X\beta_1^2 + X^2\beta_1 + \beta_1 + X^2 + X)\right. \\ & \qquad \left. + M_1M_4(X^2\beta_1^2 + X\beta_1^2 + X^3\beta_1 + X\beta_1 + X^2 + X)\right]\beta_2 \\ & + X(M_2M_3 + M_1M_4)(\beta_1 + 1)(1 + \beta_1 X). \end{aligned}$$

This equation can be solved by picking random β_1 values until the resulting quadratic equation in β_2 has solution. Assuming the coefficients of this equation behave reasonably randomly, a solution will be found after 2 trials for β_1 in mean.

6.4.3 Runtime analysis

Combining the estimations of Section 6.4.1 and 6.4.2, our preimage finding algorithm for Morgenstern hash function runs in probabilistic polynomial time.

6.5 Discussion

In this chapter, we have presented efficient algorithms finding preimages for the LPS hash function and finding collisions and preimages for the Morgenstern hash function with $q = 2$. Similar algorithms with the same complexity can be derived for the Morgenstern hash function with different q values. Our algorithms build upon the Tillich and Zémor algorithm [259] although they are not trivial extensions of it.

Our algorithms may have applications outside the cryptographic community. The preimage finding algorithm for LPS hash actually solves the diophantine Equation (6.4) which at first sight seems to be a very hard problem. The path-finding and cycle-finding algorithms may improve the understanding of LPS and Morgenstern graphs when considering their Ihara Zeta-function. Finally, some of the numerous applications of expander graphs in computer science could benefit from our new path-finding algorithms.

Because of all these actual and potential applications, we stress that our algorithms and their running time estimates still may and should be improved in many ways. The algorithms of Section 6.2 and 6.4 give paths of length about $8 \log p$ and $8 \deg P_n(X)$ while the diameter of LPS and Morgenstern graphs are known to be $2 \log p$ and $2 \deg P_n(X)$. Choosing a smaller k value in the algorithms will decrease these lengths and may also improve the running times. Finding other decompositions with less than 4 diagonal matrices is another interesting approach. Finally, adapting our algorithms to make them deterministic is a particularly interesting open problem.

Tillich and Zémor proposed to repair the LPS hash function by replacing each generator s_i by its square $s_i' := s_i^2$ [259]. With this modified hash function, the lifting strategy would require lifting the identity (or any other matrix) onto a matrix of Ω that has a very special and rare factorization in terms of the s_i (namely every factor is repeated twice) or finding a good characterization of the modified set $\Omega' := \langle s_0', s_1', ..., s_l' \rangle$. Alternatively, Tillich and Zémor also proposed to remove one of the graph generators of the LPS hash function: again, the lifting strategy would fall upon the problem of finding very special elements of Ω, belonging to a set that seems hard to characterize. Similarly, the Morgenstern hash function may be repaired by considering the directed Cayley graph generated by $s_0' := s_0 s_1$ and $s_1' := s_0 s_2$ [204]. We point out that with these modifications, the Cayley graphs subjacent to LPS and Morgenstern hash functions are no longer Ramanujan.

The main two motivations in considering LPS and Morgenstern hash functions instead of the Zémor-Tillich hash function were the Ramanujan property and the ability to process more bits at once. We already pointed out in Section 4.4 that the "optimal expansion rate" of Ramanujan graphs may be inferior to the expansion rate of well-chosen *directed* graphs. In particular, directed Cayley graphs of finite special linear groups like the Zémor-Tillich graphs tend to have large expansion rates.

Our implementations of LPS and Morgenstern hash functions (see Section 4.3) have shown that the Zémor-Tillich hash function is actually twice as fast as LPS and Morgenstern hash functions. It is easy to check that the above modifications of these algorithms will certainly be even slower, and it is not clear that a bound on their expansion rates can be derived more easily than for the Zémor-Tillich hash function.

In this chapter, we have shown that the strong mathematical structure present in the LPS and Morgenstern graph constructions can be exploited to build very efficient collision and preimage attacks against the corresponding hash functions. The same structure seems necessary to prove the Ramanujan property from the Ramanujan's conjecture [167] and it is therefore tempting to conclude that using Ramanujan graphs for expander hashes is a bad idea in general [259]. However, the Pizer hash function that we now discuss contradicts this intuition: it is also built on a Ramanujan graph family and it has resisted all attacks so far.

Chapter 7

The Pizer hash function

The Pizer hash function was introduced by Charles et al. together with the LPS hash function [68]. Besides our vectorial and projective versions of the Zémor-Tillich hash function (Section 5.5), it is the only expander hash proposal made so far that is not a Cayley graph. The function is based on the Pizer Ramanujan graph family [210]; it has been considered with a lot of curiosity by cryptographers as a new beautiful application of elliptic curves in cryptography. The function was also advertized outside the cryptographic community thanks to Mackenzie's article in *Science* journal [174].

The security of Pizer hashes relies on the hardness of building isogenies or pairs of isogenies between supersingular elliptic curves. These problems are non-standard problems in cryptography; they were partially considered before by Galbraith [109] who pointed out their use for solving elliptic curve discrete logarithms. The best known algorithms, provided by Galbraith, are exponential.

Pizer hash security also relates to representation problems in quaternion algebras. The supersingular elliptic curves over a finite field are in one-to-one correspondence with the maximal orders in a quaternion algebra related to this field: the original description of Pizer graphs was actually in the language of maximal orders in a quaternion algebra. As the correspondence itself is hard to compute, solving the representation problem does not suffice to solve the problems on isogenies.

Pizer hashes are much less efficient than other expander hashes. The efficiency is maximal when the parameter l is set to 2, but it is still more than 100 times slower than the Zémor-Tillich hash function. Indeed, a quadratic equation over $\mathbb{F}_{p^2}$ must be solved at each step while only XORs and SHIFTs must be performed for the Zémor-Tillich hash function.

In this chapter, we do not claim any significant result on the security nor the efficiency of the Pizer hash function. For the completeness of the thesis, we reproduce here some of the results present in Charles et al.'s paper and we evaluate the practicability of the function. We also point out (this is joint work with Jean-Pierre Tignol) that the function would probably be broken if it were described in the language of maximal orders rather than in the language of supersingular elliptic curves. The Chapter is organized as follows: Section 7.1 describes the Pizer hash function, Section 7.2 discusses its security and Section 7.3 its efficiency. A short conclusion is given in Section 7.4.

7.1 Description

Pizer hashes are expander hashes constructed from the Pizer Ramanujan graph family [210] previously discovered by Mestre [180]. They are parameterized by a large prime p congruent to 1 modulo 12, a small prime l and a supersingular elliptic curve over $\mathbb{F}_p$ defining the starting point in the graph.

Pizer graphs may be described in two equivalent languages but choosing one or the other may affect the efficiency and the security of the Pizer hashes. In Pizer's original paper, the vertices of the graph are maximal orders in the quaternion algebra over $\mathbb{Q}$ that is ramified only at p and ∞. In Charles et al.'s paper [68], the vertices are seen as supersingular elliptic curves over $\mathbb{F}_p$. There exists a one-to-one correspondence between both types of objects but as this correspondence has been itself hard to compute [68, 66], dealing with supersingular elliptic curves rather than with orders in quaternion algebras is more than an aesthetic choice. As we will discuss later in this chapter, it may affect positively the security and badly the efficiency.

Pizer hashes are defined as follows. Let p be a large prime congruent to 1 modulo 12 and let l be a small prime; Charles et al. propose to use $p \approx 2^{256}$ and focus mainly on $l = 2$. The set of vertices of the graph is the set V of supersingular j-invariants over the finite field $\mathbb{F}_{p^2}$; for $p \equiv 1 \bmod 12$ there are $\lfloor \frac{p}{12} \rfloor$ such points [248]. There is an edge in the graph from j_1 to j_2 if and only if there is an isogeny of degree l from E_1 to E_2, where E_1 and E_2 satisfy $j_1 = j(E_1)$ and $j_2 = j(E_2)$. When $p \equiv 1 \bmod 12$ this gives rise to an undirected $l + 1$-regular graph. The edges must be ordered in some way to complete the definition of Pizer hashes; a method is described in [68]. More details on the hash computation are given in Section 7.3 below.

Charles et al.'s paper does not fully specify the key generation algorithm for Pizer hashes but it does provide rationales for identifying sets of parameters that should be avoided. Without influencing the security, we can consider that the starting edge is fixed by the hash algorithm once the starting point is given. We can also consider that different l values define different hash functions hence the parameter l is also part of the hash algorithm and not of the key. A Pizer hash key is therefore made of a prime p congruent to 1 modulo 12 and of a supersingular j-invariant j_0 in $\mathbb{F}_{p^2}$ used as starting point in the graph.

The parameter p and the starting point may greatly influence the collision resistance of the hash function, and it is not clear to us whether an efficient key generation algorithm can actually be constructed for Pizer hashes. Pizer graphs do not have large girth in general; Charles et al. describe a method for choosing p such that the graph has no cycle of length 2. The method can be extended to prevent larger cycles but it is not clear whether the resulting key generation algorithm is efficient enough for practice. Charles et al. also suggest to choose the initial point to prevent short cycles from starting from that point. As an example, for $p \equiv 1 \bmod 24$ and $l \equiv 3 \bmod 4$ they identify a maximal order in the quaternion algebra ramified at p and ∞ such that no cycle in the Pizer graph starts from this order. As the correspondence between supersingular elliptic curves and maximal orders is not efficiently computable, it is not clear either whether this method leads to an efficient key generation algorithm including an initial supersingular j-invariant.

7.2 Security considerations

When l is fixed and p varies, Pizer graphs form a Ramanujan family of $l+1$-regular graphs, hence their non-backtracking mixing rate $\widetilde{\rho}$ is at most $\frac{1}{\sqrt{q}}$ which amounts to $\frac{1}{\sqrt{2}}$ per bit of message. Pizer's graphs have a small diameter $D(\Pi_{l,p}) \leq 2\log_l \frac{p}{12} + 2\log_l 2 + 1$ [210]. However, as discussed above, they do not have large girth unless additional restrictions are put on the prime p.

Collision and preimage resistances are implied by the hardness of problems of constructing isogenies between supersingular elliptic curves.

Problem 7.1 *Produce a pair of supersingular elliptic curves over $\mathbb{F}_{p^2}$, E_1 and E_2, and two* distinct *isogenies of degree l^{μ} and $l^{\mu'}$ between them that have a cyclic kernel, for some integers μ, μ'.*

Problem 7.2 *Given E, a supersingular elliptic curve over $\mathbb{F}_{p^2}$, find an endomorphism $f : E \to E$ of degree $l^{2\mu}$ for some integer μ that is not any automorphism of E composed with the multiplication by l^{μ} map.*

Problem 7.3 *Given E_1 and E_2, two supersingular elliptic curves over $\mathbb{F}_{p^2}$, find an isogeny $f : E_1 \to E_2$ of degree l^{μ} between them, for some integer μ.*

Finding a collision for the Pizer hash function (with a given p and a random starting point j_0) implies a solution to Problem 7.1 with E_1 such that $j(E_1) = j_0$, and a solution to Problem 7.2 with E_1 such that $j(E_1) = j_0$. Moreover, finding preimages implies a solution to Problem 7.3 with $j(E_1) = j_0$ [68]. Problems 7.1 to 7.3 are in a sense even harder than collision and preimage problems: given an isogeny between two (possibly identical) elliptic curves, decomposing this isogeny into a sequence of l-degree isogenies seems to be a hard problem itself. (However, as a large degree isogeny is a rational mapping with very large degrees in the nominator and the denominator, the only way to provide it seems to be in factorized form, in which case the problems are clearly equivalent.)

Problem 7.3 was introduced by Galbraith [109] who provided an algorithm running in time $O(p \log p)$ for supersingular elliptic curves. For ordinary curves, the problem was motivated by solving elliptic curve discrete logarithm problems. (In supersingular curves, discrete logarithm problems are usually much easier using efficiently computable bilinear maps to finite fields.) For solving Problems 7.1 and 7.2, the most natural approach would be to work with the quaternion algebra whose maximal orders are the endomorphism rings of the supersingular elliptic curves over $\mathbb{F}_{p^2}$.

Given a prime p, there exists one quaternion algebra

$$\mathbb{A}_{p,\infty} = (\alpha, \beta) := \mathbb{Q} + \mathbb{Q}\alpha + \mathbb{Q}\beta + \mathbb{Q}\alpha\beta$$

with $\alpha^2, \beta^2 \in \mathbb{Q}$, $\alpha^2 < 0$, $\beta^2 < 0$ and $\alpha\beta = -\beta\alpha$ that is ramified exactly at p and ∞. The maximal orders in this algebra are exactly the endomorphism rings of the supersingular elliptic curves over $\mathbb{F}_{p^2}$. In his paper [210], Pizer defined his graphs as follows. Each vertex corresponds to a maximal order in $\mathbb{A}$ and two orders o_1 and o_2 are neighbors if they are conjugated by an element with quaternion norm l:

$$o_2 = s_i^{-1} o_1 s_i, \qquad n(s_i) = l.$$

The elements with norm l are easy to characterize by using the equivalence between quaternions and 2×2 matrices with determinant 1. The corresponding matrices s_i are all the matrices in the set

$$S := \left\{ \begin{pmatrix} 1 & b \\ 0 & l \end{pmatrix} | 0 \leq b \leq l-1 \right\} \bigcup \left\{ \begin{pmatrix} l & 0 \\ 0 & 1 \end{pmatrix} \right\}.$$

The set of matrices generated by products of S with length μ is therefore

$$S^{(\mu)} := \left\{ \begin{pmatrix} l^e & b \\ 0 & l^{\mu-e} \end{pmatrix} | 0 \leq e \leq \mu, b < l^{\mu-e} \right\}.$$

Given a starting element in $\mathbb{A}$, it is therefore possible to write an equation expressing the fact that the conjugation of this element by an element of $S^{(\mu)}$ leads to a collision, and solving this equation brings a cycle in Pizer graphs.

This analysis is inspired by the attacks against LPS and Morgenstern hash functions (see Chapter 6) and it is made possible by the very special structure of the matrices involved. This analysis would probably lead to collisions against the Pizer hash function if it was described in the language of orders in a quaternion algebra like in Pizer's paper [210]. However, as the Pizer hash function is described in the language of elliptic curves and the correspondence between the two "worlds" is hard to compute, the Pizer hash remains currently unbroken.

7.3 Efficiency considerations

We now give some details on Pizer hash computation, that is, we show how to perform a walk from one vertex to one of its neighbors according to the message's digits. The description follows [68].

Suppose we are at a given vertex labeled by a supersingular j-invariant j. To this j-invariant corresponds an elliptic curve with equation $E(j)$: $y^2 = x^3 + 3kx + 2k$ where $k = \frac{j}{j-1728}$. Let $E[l]$ be the l-torsion of E; it is a subgroup of E that is isomorphic to $\mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z}$. The l-torsion has $l + 1$ non-trivial cyclic subgroups, each of them defined by one of its generators. By calculating the whole l-torsion, it is possible to fix a canonical ordering of the $l + 1$ subgroups $H_0, ..., H_l$ [68]. To each subgroup H_i of the l-torsion corresponds an isogeny of degree l that can be computed with Vélu's formulae [262, 68]; the isogeny maps E to E/H_i and j to its corresponding neighbor in the graph.

Pizer hash computation is not very efficient. Vélu's formulae only involve additions, multiplications and squarings but computing the l-torsion of the curve is the bottleneck at each step of the hash computation as it essentially requires computing all the roots of the modular polynomial of order l which can have a degree up to $(l + 1)^2$. When $l = 2$, this step requires solving a quadratic equation. Charles et al. evaluate that the whole algorithm takes the time of about $2\log(p)$ multiplications per step. This is not very efficient with respect to Zémor-Tillich, LPS and Morgenstern computation that only require a few additions. On a 64-bit AMD Opteron 252 2.6GHz, Charles et al.'s C implementation has a throughput of 13.1kb/s for primes p of 256 bits: this corresponds to 1.588M cycles/byte!

The efficiency of Pizer hashes may probably be improved for $l = 2$ by choosing primes p with a particular form and in general with a clever use of the modular polynomial properties [223] or maybe by changing the curve coordinates. However, it will certainly never reach throughputs comparable to SHA or even to Zémor-Tillich. In the language of quaternions or matrices, the computation would only require a few multiplications per step, but as pointed out above it would also be much less secure.

7.4 Conclusion

Pizer hash functions are an interesting new beautiful use of elliptic curves in cryptography. Unlike the other expander hash functions considered in this thesis, their security does not depend on the hardness of a representation problem but on isogeny problems and the hardness of translating these problems in the language of quaternion orders. As these isogeny problems are also somehow related to elliptic curve discrete logarithm problems [109], Pizer hash function might be the most secure of all expander hash functions so far for well-chosen parameters. However, Pizer hashes will probably never move from an interesting theoretical construction to a function actually used in practice. Indeed, the hash computation is rather slow and complex and it is not clear whether an efficient key generation algorithm may be found.

Part III

Perspectives

Chapter 8

Non-Malleability Property for Hash functions

Expander hash functions are *malleable*. Given the hash value of a message m it is possible to construct the hash value of a message that is related to the previous one in some particular way. It is also possible to produce two messages such that their hash values satisfy some particular relation. Those properties do not necessarily contradict preimage nor collision resistance of expander hashes in general, but they may induce security issues if they are used in protocols that require hash functions with sufficient pseudorandomness properties.

The security consequences of expander hash malleability properties differ in applications. Collision resistance is a sufficient property when the hash function is only used to compress data. On the other hand, if the hash is used to destroy some mathematical structure (either known or unknown) like in hash-then-sign or Fiat-Shamir signatures, security proofs typically take place in the random oracle model. Security in the standard model requires further (stronger) assumptions on the hash function and/or the other protocol's components, and may even require to modify the protocol, most often at the cost of some efficiency.

Setting apart efficiency, collision resistant hash functions are sufficient to build signature schemes and commitment schemes but not message authentication codes and pseudorandom number generators.

The cryptographic community has been searching for good security definitions to replace the random oracle model, some of which are useful to understand what properties expander hashes fail to satisfy. Two notions of the literature are particularly relevant here, correlation intractable hash functions and non-malleable hash functions. *Correlation intractable hash* *functions* [63] generalize collision resistance: not only should it be hard to find two messages whose hash values are equal, but also to find two messages whose hash values satisfy another relation.

The notion of *non-malleability* is well-understood in the context of encryption, commitments and zero-knowledge since the pioneering work of Dolev et al. [94]. For hash functions, it has only recently received a formal treatment by Boldyreva et al. [50]. Given a non-malleable hash function, it is hard to construct the hash values of two related messages from one of these hash values. In a sense, if a hash function is not correlation intractable it is possible to manipulate its outputs while if it is not malleable it is possible to manipulate its inputs. In both cases, it makes sense to parameterize the definition by a class of relations, specifying which manipulation we want to prevent with the definition.

In many protocols that only have a proof in the random oracle model, correlation intractability or non-malleability requirements are easily seen to be necessary. In some protocols like RSA signatures they also *seem* sufficient but providing a meaningful definition of non-malleable hash function that would give security guarantee for RSA signatures remains an open problem today.

The malleability properties of a hash function, and in particular of expander hashes, may also prove useful if exploited cleverly. For instance, Quisquater and Joye [221] used the associativity of the Zémor-Tillich hash function in a protocol identifying video sequences and Lyubashevsky and Micciancio [171] used the linearity of SWIFFT to build an asymptotically efficient digital signature scheme. Some applications of the more requiring incremental hash functions [35, 36, 38, 209] also generalize to Cayley hashes.

At the light of existing provably secure hash functions (including expander hashes), malleability seems to be a necessary price for enjoying an efficient "proof" of collision resistance. Number theoretic and other mathematical problems have strong algebraic structures, which unavoidably induce some malleability properties. Reasonably non-malleable hash functions can certainly be constructed with pseudorandom number generators, which can be based on one-way functions and hence on number-theoretic problems. However, the resulting constructions are very inefficient. As often in Cryptology, efficiency and security seem to be contradictory here.

Perfectly one-way hash functions [61, 64] are a first step towards designing all-purpose hash functions from collision resistant hash functions. However, these functions can still be malleable and they are probabilistic, which may lead to some practical issues. The non-malleable hash function construction of [50] uses a perfectly one-way hash function and a simulation-sound non-interactive zero-knowledge proof of knowledge, hence it requires some "global source of randomness" which again presents practical issues.

As efficient, all-purpose secure hash functions seem hard to design, other approaches are possible. Anderson [27] has stressed that protocol designers should be more explicit on the properties they require for the hash functions they use (rather than just modeling the hash function as a random oracle). Some protocols may only require collision resistance and non-malleability with respect to additive relations, in which case the Zémor-Tillich hash function could be a safe choice. In Chapter 9, we will present a semi-provable hash function based on Zémor-Tillich, which is provably collision resistant but which also (heuristically) satisfies pseudorandomness properties as good as dedicated hash functions like SHA.

This chapter examines the positive and negative consequences of the malleability properties of expander hashes. We do not claim any original contribution here, but rather an interesting and necessary literature review around the applicability of malleable hash functions in cryptography. We focus on expander hashes of course, but most observations here are also relevant to other provable hash functions.

In Section 8.1, we describe the malleability properties of expander hashes and we identify the applications that can or cannot use expander hashes. In Section 8.2, we give correlation intractability and non-malleability definitions and show that they can help capturing the malleability of expander and Cayley hashes. In Section 8.3, we show how malleability properties can be exploited as an (efficiency) advantage and we conclude the chapter in Section 8.4 .

8.1 (In)security of protocols with malleable hash functions

8.1.1 Malleability of expander and Cayley hashes

We briefly recall the malleability properties of expander and Cayley hashes described in Section 4.2.7. By construction, for any messages m, m' it is possible to compute the hash value of $m||m'$ from the hash value of m without even knowing m, by using the neighbor ordering function. Moreover for Cayley hashes, the associativity of the group's law implies that for any initial point g_0 and messages m and m', we have

$$H(m||m') = H(m) \cdot g_0^{-1} \cdot H(m')$$

where $\cdot$ represents the group operation. In particular, if g_0 is the identity then

$$H(m||m') = H(m) \cdot H(m').$$

8.1.2 Insecure protocol-hash associations

Cryptographic schemes using hash functions as a source of pseudorandomness should not be implemented with a malleable hash function. Key derivation and pseudorandom number generation algorithms that are secure in the random oracle model clearly become insecure with a malleable hash function. An auction protocol may be implemented in the random oracle as follows: each auctioneer commits to a price m by revealing $H(m)$; after everybody has committed, the prices are revealed by the auctioneers. Now suppose that only two auctioneers are competing. If a Cayley hash is used and the first auctioneer has already committed by $H(m)$, the second auctioneer can propose $H(1||m)$: he is then ensured to win the auction (although he does not even know the price $1||m$ he has committed to before the first auctioneer reveals his price).

Anderson first observed that many protocols are insecure with hash functions that have some malleability properties, even if they are collision resistant [27]. As simplest examples, he introduced the notions of complementation resistance, addition resistance and multiplication resistance. A hash function H is complementation resistant if it is hard to produce two messages m and m' such that $H(m) \oplus H(m')$, it is addition resistant if it is hard to produce three messages m_1, m_2, m_3 such that $H(m_1) + H(m_2) = H(m_3)$, and it is multiplication resistant if it is hard to produce three messages m_1, m_2, m_3 such that $H(m_1) \cdot H(m_2) = H(m_3)$. Complementation freedom hash functions may be useful in schemes built on DES algorithm [9]; multiplication freedom is definitely necessary in multiplicative homomorphic schemes like RSA signatures (see Section 8.1.4). Similarly, many other cryptographic schemes (Schnorr, DSS,...) implicitly rely on non-classical assumptions on the hash functions they use. The Fiat-Shamir transform (Section 2.6.2) is not sound in general: there exists a three round authentication scheme, which does not give a secure signature scheme when Fiat-Shamir is applied to it with any efficient hash-function [119].

8.1.3 Secure protocol-hash associations

If many protocols rely on the random oracle model, there exist also many protocols that only need a universal hash function and/or a collision resistant hash function. As mentioned in Section 2.6.2, signature schemes can be constructed from one-time signatures schemes hence from collision resistant hash functions and universal one-way hash functions. The signature scheme in [199] relies on a particular non-malleability assumption on the hash function; the commitment scheme in [127] needs a collision resistant hash function and an universal hash function. The encryption scheme of Bellare and Rogaway [41] is secure with perfectly one-way hash functions [61]. HMAC is a PRF if its compression function is a PRF [32]. Alternatively, Fischlin [102] showed that HMAC and NMAC are secure MAC (see Definition 2.12) if the compression function is non-malleable in the sense of Definition 8.3 below.

8.1.4 An open problem: full-domain RSA signatures

Among protocols that are secure in the random oracle model, some are insecure in the standard model [63, 119] and others are secure under additional assumptions on the hash function [199, 127, 32, 102]. However, the security of many protocols among the most standard ones remains an open problem. In this section, we briefly discuss the case of full-domain RSA signatures, whose security in the standard model has been challenging cryptographers for years.

Full-domain RSA signatures are constructed from RSA signatures with the hash-then-sign paradigm. In RSA signatures, the key generation algorithm produces an RSA modulus n and a pair (d, e) such that $de = 1 \bmod \varphi(n)$. The public key is (n, e) and the private key is d. Given a message $m \in [2, n-2]$, its signature is $\sigma := m^d \bmod n$, and the signature verification amounts to checking whether $\sigma^e = m$. Given a message $m \in \{0,1\}^*$, a hash function H and its key s, the full-domain RSA signature of m is $\sigma := H(s,m)^d$.

In this signature scheme, the hash function is used to extend the domain but also to destroy the algebraic structure of the RSA group $\mathbb{Z}_n^*$. It is easy to list properties of the hash function that are necessary for the security of RSA signatures, but on the other hand it seems very hard to prove that these properties are sufficient. Intuitively, the full-domain RSA signature scheme looks as follows to the attacker:

$$m \longrightarrow H(s,m) = \sigma^e \longleftarrow \sigma = H(s,m)^d.$$

The scheme seems secure because both computing e-roots modulo n and inverting the hash function are hard problems: to produce a forgery $(m, \sigma = H(s,m)^d)$ an attacker seems to be required to solve the first problem if he starts from a message and the second problem if he starts from a signature.

The homomorphic properties of modular exponentiation make things harder. Even if it is hard to compute e-roots in general, the e-roots of 0 and 1 are 0 and 1 and it is possible to combine various e-roots into a new one: if $h_i = \sigma_i^e$ then for any integers α_i, $(\prod h_i^{\alpha_i}) = (\prod \sigma_i^{\alpha_i})^e$. If no additional condition is put on the hash function, the scheme looks more as follows to the attacker:

$$\begin{tikzcd} m \arrow[r] & H(s,m) = \sigma^e \arrow[r, dashed, shift right] & \sigma = H(s,m)^d. \arrow[l, shift right] \end{tikzcd}$$

To prove the security of full-domain RSA signatures, additional conditions are required both on the hash function and on the RSA group. At least, preimages of 0 and 1 must be hard to compute and it must be hard to compute m_i and α_i such that $h_i := H(s, m_i)$ satisfy $\prod h_i^{\alpha_i} = 1$. In this case, all the strategies exploiting the evident algebraic structure of the RSA group would fail and the attacker's view of the forgery problem would look as follows:

$$\begin{tikzcd} m \arrow[r, shift left] & H(s,m) = \sigma^e \arrow[l, dashed, shift left] \arrow[r, dashed, shift right] & \sigma = H(s,m)^d. \arrow[l, shift right] \end{tikzcd}$$

Now, no trivial breaking strategy would be successful but there may exist other attacks. It seems necessary to require that there exists no other way to "go from the middle to the right", an hypothesis that can be formalized by a "knowledge of exponent" type assumption on the RSA group [125, 39].

When we keep working with informal definitions, this set of assumptions seems sufficient for the security of full-domain RSA signatures. However, formalizing these intuitions proved to be a difficult task and we had to leave it as an open problem. The first issue we faced was of course to provide meaningful (non)-malleability definitions.

8.2 Non-malleability definitions

Providing good definitions of non-malleable hash functions is an important problem both to prove the security of many protocols in the standard model and to characterize which properties expander hashes fail to satisfy. Two definitions from the literature are especially relevant: correlation intractability [63] and non-malleability [50].

8.2.1 Canetti et al.'s correlation intractability

Correlation intractability appeared in the milestone paper where Canetti et al. [63] proved that random oracles do not exist; it extends a definition of Okamoto [199] and was informally sketched by Anderson [27]. Correlation intractability extends the notion of collision resistance; it is defined with respect to *evasive* relations, which are relations between inputs and outputs of a random oracle which are satisfied only with a negligible probability.

Definition 8.1 *Let $l_{in}, l_{out} : \mathbb{N} \rightarrow \mathbb{N}$ be length functions of the security parameter. A relation $R : (\{0,1\}^{l_{in}} \times \{0,1\}^{l_{out}})^N \rightarrow \{0,1\}$ over N input-output sequences is* evasive *with respect to (l_{in}, l_{out}) if for any* `PPT` *algorithm A, the probability*

$$Adv_R^{\mathit{Ev},A}(n) := \Pr\left[Exp_R^{\mathit{Ev},A}(n) = 1\right]$$

is negligible, where $Exp_R^{\mathit{Ev},A}$ is defined below.

Experiment $Exp_R^{\mathtt{Ev},A}(n)$:

- a function $\mathcal{O} : \{0,1\}^{l_{in}(n)} \rightarrow \{0,1\}^{l_{out}(n)}$ is randomly selected;
- the adversary A has an oracle access to $\mathcal{O}$: he sends queries $m_i \in \{0,1\}^{l_{in}}$ of his choice and receives the corresponding $\mathcal{O}(m_i)$;
- the adversary A outputs a tuple $(m_1, ..., m_N) \in (\{0,1\}^{l_{in}(n)})^N$;
- $Exp_R^{\mathtt{Ev},A} = 1$ if $R(m_1, \mathcal{O}(m_1), ..., m_N, \mathcal{O}(m_N)) = 1$.

A hash function is correlation intractable with respect to an invasive relation R if it is hard to compute inputs that together with their corresponding hash values satisfy the relation.

Definition 8.2 *A hash function $\mathcal{H} := (Gen, H)$ is correlation intractable with respect to an invasive relation $R : (\{0,1\}^{l_{in}} \times \{0,1\}^{l_{out}})^N \rightarrow \{0,1\}$ over N input-output sequences if for any* `PPT` *algorithm A, the probability*

$$Adv_{\mathcal{H}}^{\mathit{CI},R,A} := \Pr\left[Exp_{\mathcal{H}}^{\mathit{CI},R,A} = 1\right]$$

is negligible, where $Exp_{\mathcal{H}}^{\mathit{CI},R,A}$ is defined below.

Experiment $Exp_{\mathcal{H}}^{\texttt{CI},R,A}$:

- a key s is generated by running Gen on input 1^n;
- the key s is given to the adversary A;
- the adversary A outputs a tuple $(m_1, ..., m_N) \in (\{0,1\}^{l_{in}(n)})^N$;
- $Exp_{\mathcal{H}}^{\texttt{CI},R,A} = 1$ if $R(m_1, H(s, m_1), ..., m_N, H(s, m_N)) = 1$.

The notion is limited to evasive relations to discard relations that can even be satisfied with random oracles. A hash function is correlation intractable if it is correlation intractable with respect to all invasive relations. Canetti et al. [63] showed that random oracles are correlation intractable hash functions and that correlation intractable hash functions do not exist, therefore showing that random oracles do not exist.

Definition 8.2 is meaningful when it is restricted to particular relations. For example, Okamoto [199] proved the security of a signature scheme based on correlation intractability with respect to multiplicative relations. The malleability of expander hashes is captured by relations

$$R(m_1, h_1, m_2, h_2) = 1 \Leftrightarrow (m_2 = m_1 || i) \wedge (h_2 = \theta(h_1, i))$$

where θ is the neighbor ordering function (Section 4.2.1). The malleability of Cayley hashes is captured by more general classes of relations, for example

$$R(m_1, h_1, m_2, h_2, m_3, h_3) = 1 \Leftrightarrow (m_3 = m_1 || m_2) \wedge (h_3 = h_1 \cdot h_2)$$

where $\cdot$ represents the group operator.

8.2.2 Boldyreva et al.'s non-malleability

Another non-malleability definition for hash functions has recently been proposed by Boldyreva, Cash, Fischlin and Warinschi [51, 101, 102]. The definition is in the vein of non-malleability definitions for encryption, commitment and zero-knowledge proofs of knowledge [94]. It involves a simulator acting in an idealized experiment. Like in perfectly one-way hash functions, a hash function here is a triple of `PPT` algorithms $\mathcal{H} = (Gen, H, V)$, the hash algorithm is probabilistic, and there is a verification algorithm.

Definition 8.3 *A hash function $\mathcal{H} = (Gen, H, V)$ is called non-adaptive, single-value non-malleable (with respect to the parameterized distribution χ, function* HINT *and a relation $\mathcal{R}$) if for any* PPT *algorithm $\mathcal{A}$ there exists a* PPT *algorithm $\mathcal{S}$ such that for any relation $R \in \mathcal{R}$ the difference*

$$\Pr[Exp_{\mathcal{H}}^{\textit{nmh-1},A}(n) = 1] - \Pr[Exp_{\mathcal{H}}^{\textit{nmh-0},A}(n) = 1]$$

is negligible, where $Exp_{\mathcal{H}}^{\textit{nmh-1},A}(n)$ and $Exp_{\mathcal{H}}^{\textit{nmh-0},A}(n)$ are defined below.

Experiment $Exp_{\mathcal{H}}^{\texttt{nmh-1},A}(n)$:	Experiment $Exp_{\mathcal{H}}^{\texttt{nmh-0},A}(n)$:
- a key s is generated by running Gen on 1^n;	- a key s is generated by running Gen on 1^n;
- a message m is generated by running χ on 1^n;	- a message m is generated by running χ on 1^n;
- a hint h_s on m is generated by running HINT on 1^n and m;	- a hint h_s on m is generated by running HINT on 1^n and m;
- the value $h := H(s, m)$ is computed;	
- the values s, h and h_s are given to the adversary A;	- the values s and h_s are given to the simulator S;
- the adversary returns a function $T : \{0,1\}^* \to \{0,1\}^*$ and a hash value h^*;	
- the value $m^* = T(m)$ is computed;	- the simulator returns a message m^*
- $Exp_{\mathcal{H}}^{\texttt{nmh-1},A}(n) = 1$ iff $R(m, m^*) = 1$ and $h^* \neq h$ and $V(s, x^*, y^*) = 1$.	- $Exp_{\mathcal{H}}^{\texttt{nmh-0},A}(n) = 1$ iff $R(m, m^*) = 1$.

From the hash value of a message m, but without this message, the adversary A is required to provide the hash value of a message m^* that is related to m. The definition is parameterized by a distribution χ on the hash function inputs reflecting the fact that the message may not follow the uniform distribution, a hint algorithm revealing some side-information about the message m and a relation R. Definition 8.3 can be adapted to compression functions [102] and extended to adaptive adversaries seeing more hash values [50].

Non-malleable hash functions can be constructed for a large class of relations using perfectly one-way hash functions and simulation-sound non-interactive zero-knowledge proofs of knowledge but they cannot be reduced to one-way functions. The notion implies perfect one-wayness but it does not imply one-wayness nor collision resistance [51].

The definition captures the malleability of expander hashes via the relation

$$R(m, m^*) = 1 \Leftrightarrow \exists m' \text{ s.t. } m^* = m||m'$$

and the malleability of Cayley hashes through the relation

$$R(m_1, m_2, m^*) = 1 \Leftrightarrow m^* = m_1||m_2.$$

8.3 Positive uses of malleable hash functions

The expander hash design has many interesting properties but it also induces inherent malleability. Section 8.1 has presented the security issues following from the malleability of hash functions. In this section, we describe positive applications of three kinds of malleable hash functions: the lattice-based FFT hash function of Lyubashevsky et al. [172, 173] (see Section 3.3), incremental hash functions and Cayley hash functions. Incremental hash functions [35, 36, 38, 209] are hash functions with an update algorithm $Incr$ allowing the update of a hash function if some message block is modified, with an update cost ideally proportional to the amount of modification in the message.

The linearity of the FFT hash was used in [171] to construct a one-time signature scheme. If H is the FFT hash, the key generation algorithm generates a key s for the hash function and two values x, y. The public key of the scheme is $(s, H(s,x), H(s,y))$, its secret key is (x, y), the signature of a message m is $\sigma := x \cdot m + y$ and this signature is checked by verifying that $H(s,\sigma) = H(s,x) \cdot m + H(s,y)$. The signature scheme is provably secure and asymptotically efficient [171].

Incremental hash functions were introduced in [35] for authenticating messages that are sent to various persons with different headers, for re-authentication of large data disks after small modifications to provide virus protection, or for authenticating videos or more generally data that only slightly changes with time. Expander and Cayley hashes are incremental hash functions restricted to some kinds of modifications. The first application above may also use Cayley hashes and some non-Cayley expander hashes; the second can certainly use Cayley hashes if partial hash values are stored (this is done in [221] for the Zémor-Tillich hash function); the third application however requires malleability properties that Cayley hash do not possess.

The associativity of group laws makes parallel computation of Cayley hashes particularly easy and efficient. Assuming long messages $m_0 ... m_\mu$ have to be hashed and N computing units are available, the messages can be decomposed into message blocks of length μ/N. Each block can then be given to a different unit; the first unit computes $H(m_0 ... m_{\mu/N})$ and the unit i computes $g_0^{-1}H(m_{(i-1)\mu/N} ... m_{i\mu/N-1})$. When all units are done, the hash value of the full message may be computed with only $N-1$ group operations.

The interest of this property for hashing long messages cannot be overstated. Hardware computing machines and current and future multi-core processors decrease computing time by adding parallel computing units. Merkle-Damgård-based hash functions including the SHA algorithms are not suited to parallel architectures as they treat message blocks sequentially. Cayley hashes support any degree of parallelism and are therefore well-prepared to future changes in computer architectures.

Cayley hashes can be computed both sequentially and in parallel, a property that really makes them unique. We point out that if many hash functions submitted to the SHA-3 NIST's competition propose a parallel mode of computation, this mode actually results in a different function: the sequential and parallel mode of computation do not give the same hash values. In some protocols (server-clients protocols, RFID protocols), one party is willing to afford some hardware expense to gain time efficiency while the other party is less concerned by computation time and more by expenses. For these specific applications, Cayley hashes would really be invaluable.

8.4 From malleable CRHF to all-purpose hash functions

Like all provably secure hash functions, expander and Cayley hash functions are malleable, a property that can be positively exploited (mainly for efficiency) but that may also induce security threats in protocols relying on more than collision resistance. Following Anderson [27], we stress that protocols designers should not just establish the security of their protocols in the random oracle model but be more explicit about the properties they require from the hash function they use. Cayley hashes are malleable but (at first sight) only with respect to some specific relations; they can *a priori* be used in protocols requiring non-malleability with respect to other relations.

Simple additional design may be sufficient to remove the malleability properties of a provably collision resistant hash function. Hash functions built following the Merkle-Damgård paradigm structurally suffer from key extension attacks. The simple HMAC construction destroys this structure and allows using them for message authentication codes. At a different level, the compression function of most dedicated hash functions is made of a round function iterated many times. The round function itself is very malleable (and even not collision resistant) but the successive iterations remove all the existing structure in the compression function.

The next chapter is dedicated to ZesT, a new hash function based on the Zémor-Tillich hash function. ZesT uses the vectorial version of ZT hash (see Section 5.5) with some additional design to remove all the existing structure but without loosing provable collision resistance and the ability to parallelize the computations. We believe that this design strategy synthesizes the main advantages of the fully heuristical and the fully theoretical strategies, and could prove useful to extend the application range of all provable hash functions.

Chapter 9

ZesT: an all-purpose hash function based on Zémor-Tillich

Provable hash functions, whose collision resistance relies on hard mathematical problems, are very appealing. Collision resistance is by far the most important property that a hash function should satisfy. A mathematical problem can be studied independently outside the cryptographic community and the confidence in the collision resistance of the hash function may increase with the understanding of the problem. However, existing provable hash functions have a rich mathematical structure implying homomorphic properties and weak behaviors on particular inputs. In general, provable hash functions should only be used in applications requiring no more than collision resistance.

On the other hand, hash functions are the Swiss army knives of cryptography: they are supposed to possess a lot of functionalities for a lot of different applications. In practice, it would be dangerous to publicize a collision resistant hash function with certain weak behaviors because the function might be (wrongly) used by engineers without any background in Cryptology. At the light of existing proposals, provable hash functions seem therefore not practical at all. However, the ZesT hash function presented is a provably collision resistant hash function that suits all applications.

ZesT is Zémor-Tillich with Enhanced Security inside. It is essentially the vectorial version of the Zémor-Tillich hash function (see Section 5.5) iterated twice. As a result, ZesT preserves the *provable* collision resistance of the Zémor-Tillich hash function but it also *heuristically* satisfies good pseudorandom properties.

We argue that this approach is meaningful as it combines the main advantages of the fully heuristical approach and the fully theoretical approach. Our function has *heuristic* pseudorandom properties comparable to the heuristic pseudorandom properties of custom designed hash functions and it is moreover *provably* secure with respect to the crucial notion of collision resistance. It is also by far more efficient than any provably pseudorandom and collision-resistant hash function could be. Besides, we point out that the same approach was chosen by Micciancio et al. in their NIST submission based on SWIFFT.

ZesT has exciting flavors of provable security. Like the Zémor-Tillich hash function, ZesT is collision resistant if and only if the balance problem is hard and in particular if the representation problem is hard for the group $SL(2, \mathbb{F}_{2^n})$ and the generators $A_0 = \left(\begin{smallmatrix} X & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and $A_1 = \left(\begin{smallmatrix} X & X+1 \\ 1 & 1 \end{smallmatrix}\right)$. According to existing attacks (see Chapter 5), ZesT is provably preimage resistant, second preimage resistant and collision resistant up to $n/2$ bits, for an output of $2n$ bits.

For collision and second preimage resistance, the security of ZesT is indeed of $n/2$ bits in the sense that attacks with this complexity exist and that attacks with lower complexity would improve the resolution of the balance problem. For preimage resistance, the actual security of ZesT seems to be as large as $2n$ bits because the preimage attacks against the Zémor-Tillich hash function do not generalize to ZesT. Besides, an informal reasoning on the known weaknesses of Zémor-Tillich tend to assert the security of ZesT as a MAC, and tests performed with Dieharder [5] provide a first positive evidence for its pseudorandom behavior.

ZesT provides great recipes for many diets: it is really practical in a wide range of applications. ZesT is provably secure, reasonably efficient in software, about as efficient as SHA in FPGA and very compact in low area ASIC implementations. ZesT algorithm only consists of XORs, SHIFTs and TEST operations on bit vectors. This simplicity allows efficient implementations on a wide range of platforms, as well as software-assisted code optimization.

At equivalent collision resistance security levels, ZesT is 10 times as slow as SHA-1, 10 times as slow as SHA-256 and 4 times as slow as SHA-512 on our 32-bit architecture evaluation platform. With the notable exception of SWIFFT [172], this is comparable or better than other provably secure hash functions, and fast enough for most software applications that are anyway limited in speed by the poor efficiency of their asymmetric cryptographic components. Moreover, the function would take full benefit of graphic accelerators with large data buses and instructions.

ZesT is particularly efficient in hardware. For high speed designs in FPGA, first evaluations show that ZesT implementations may reach throughput per slice ratios comparable to very optimized FPGA implementations of

SHA-1 and SHA-2. On the other hand, the simplicity of ZesT allows trading throughput for area with a lot of flexibility. Lightweight implementations of ZesT are much smaller than lightweight implementations of currently used hash functions. In particular, ZesT outperforms the lightweight implementation of SHA-1 presented in [99], the SQUASH implementation of [123] and the implementation of the block cipher-based hash function DM-PRESENT-80 presented in [98].

ZesT can be cut into small pieces. The function has inherited the parallelism of the Zémor-Tillich hash function. Unlike many hash functions proposed for the NIST competition, the computation of ZesT in serial and parallel modes gives the same result thanks to the inherent group structure of the function and in particular to the associativity property. Besides, additional parallelism can be exploited in software by using SIMD instructions for computing the XORs of large bit vectors.

After appropriate preparation, ZesT sweetly fits into NIST's cooking pot. Despite its very interesting properties, ZesT hash function does not completely fulfil standard requirements for hash functions like described in NIST's call [1]. However, slight modifications of the algorithm allow it to reach these requirements.

ZesT as such is a keyed hash function that possesses some weak keys, while a secure unkeyed version is necessary in many applications. This issue is solved by an appropriate choice of default keys. ZesT's collision resistance is only the square root of the birthday bound, and its second preimage resistance is not better than its collision resistance. The first issue is solved by using the projective version of Zémor-Tillich instead of the vectorial version in the second round of ZesT; and the second issue is solved by doubling the parameters' sizes in the first round. Finally, ZesT's parameter n must be prime to be protected against subgroup attacks while NIST required output lengths of 224, 256, 384 and 512 bits. This issue can be solved either by truncating or by extending the outputs by a few bits.

All these changes in ZesT's recipe may influence its simplicity, its efficiency and its security. Like tastes and colors this cannot be discussed. We believe that the choice between one or another version of ZesT's recipe will depend on everybody's personal taste for simplicity, efficiency, security and conformity with NIST's requirements. We present our favorite recipe but we invite everybody to select their best personal choice of ingredients.

This chapter presents joint original work with Giacomo de Meulenaer, Jean-Jacques Quisquater, Jean-Pierre Tillich, Nicolas Veyrat-Charvillon and Gilles Zémor. The essence of ZesT's recipe was published jointly with Nicolas Veyrat-Charvillon and Jean-Jacques Quisquater [208] together with software implementation results and a first study of pseudorandom properties. The vectorial and projective versions of Zémor-Tillich were proved secure in collaboration with Jean-Jacques Quisquater, Jean-Pierre Tillich and Gilles Zémor [207]. Hardware results were obtained with Giacomo de Meulenaer and Jean-Jacques Quisquater [88] and a paper version of this chapter will soon be published with all these people.

The chapter is organized as follows. Section 9.1 describes ZesT's key generation and hash algorithms. Sections 9.2 and 9.3 give provable and heuristic security results on ZesT. Section 9.4 gives efficiency results for a software optimized C code, a high-speed FPGA implementation and a low-area ASIC implementation and it furthermore discusses additional implementations. Section 9.5 modifies the function to approach NIST's requirements, Section 9.6 discusses some alternative choices in the design's parameters and Section 9.7 concludes the chapter.

9.1 ZesT hash function

In this section, we describe the ZesT hash function. We start by recalling the main issues in the Zémor-Tillich hash functions, then we successively present ZesT's hash algorithm and key generation algorithm.

9.1.1 Security issues with the Zémor-Tillich hash function

As discussed in Chapter 5, the Zémor-Tillich hash function is an interesting hash candidate but it has major issues preventing its use as a general purpose hash function. In particular, it is malleable (Section 4.2.7), invertible on small messages (Section 5.3.1) and it has preimage, second preimage and collision resistance security of $n/2$ bits instead of the ideal bounds of respectively $3n$, $3n$ and $3n/2$ bits.

9.1.2 ZesT hash algorithm

A binary polynomial $P_n(X)$ and a vector $\begin{pmatrix} a & b \end{pmatrix} \in \mathbb{F}_{2^n}^2$ can be represented as bit sequences of sizes n and $2n$ respectively. In this chapter, we will often abusively identify a polynomial $P_n(X) = X^n + p_{n-1}X^{n-1} + ... + p_1X + p_0$ to its corresponding bit sequence $p_{n-1}...p_1p_0$. Moreover, the elements of $\mathbb{F}_{2^n}$ can be seen as polynomials of degree less than or equal to $n-1$ once an irreducible polynomial has been fixed, hence the vector $\begin{pmatrix} a & b \end{pmatrix} =$

$\left(\begin{smallmatrix} a_{n-1}X^{n-1}+...+a_1X+a_0 & b_{n-1}X^{n-1}+...+b_1X+b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2$ will be abusively identified to the bit sequence $a_{n-1}...a_1a_0b_{n-1}...b_1b_0$.

ZEST algorithm takes as entry a key made of an irreducible binary polynomial $P_n(X)$ and of a starting point $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)$, and a bitstring $m = m_0m_1...m_{\mu-1}$ of arbitrary length. We recall from Section 5.5.1 that the vectorial Zémor-Tillich hash function on parameters $P_n(X)$ and $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right)$ is defined by

$$H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), m\right) := \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) H_{ZT}\left(P_n(X), m\right)$$

where H_{ZT} is the Zémor-Tillich hash function. The ZEST hash function is defined by

$$\text{ZEST}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), m) := H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), \left(m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), m\right)\right)\right).$$

ZEST algorithm is made of two rounds of the vectorial Zémor-Tillich hash function: after the first round, the intermediary result

$$\left(\begin{smallmatrix} a & b \end{smallmatrix} \right) := H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), m\right)$$

is seen as a bit sequence of $2n$ bits that are processed as a continuation of the message bits.

9.1.3 ZesT key generation algorithm

The key of ZEST is made of an irreducible polynomial $P_n(X)$ and of a vector $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)$. Both elements are randomly chosen by the key generation algorithm: if the polynomial is fixed, collision resistance in the sense of the definition of Section 2.2 cannot be reached: an adversary can simply store a collision for the original Zémor-Tillich hash function to produce collisions for any starting point $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right)$. On the other hand, if the starting vector is not chosen randomly, the collision resistance of ZEST is no longer equivalent to the collision resistance of the Zémor-Tillich hash function. In particular, some keys are weaker than others, for example if the starting point is $\left(\begin{smallmatrix} a & aX \end{smallmatrix} \right)$ for any $a \in \mathbb{F}_{2^n}^*$.

If the person who generates the key is trusted, the degree of the polynomial must be prime in order to avoid subgroup attacks against the Zémor-Tillich hash function, and the starting vector must be chosen randomly among all possible vectors for the reduction of Proposition 5.11 to hold. If the person generating the key is not trusted, it is necessary to choose the polynomial $P_n(X)$ and the initial vector in a way that clearly discards trapdoor attacks. This protection can be achieved by standard techniques, resorting either to universal constants like π or e, to a pseudorandom number generator or to a cryptographic hash function H (which can even be ZesT with some fixed key).

9.2 Security reduction for ZesT

ZesT has exciting flavors of provable security. Its collision, preimage and second preimage resistances follow from the hardness of the balance problem corresponding to the Zémor-Tillich hash function. In the remaining of this chapter, we assume that the complexity of solving this problem is determined by the best attack known so far which is the attack of Section 5.4.2.

9.2.1 Collision resistance

ZesT is collision resistant if and only if the balance problem corresponding to the Zémor-Tillich hash function is a hard problem.

Proposition 9.1 *There exists a* PPT *algorithm that breaks the collision resistance of* ZesT *if and only if there exists a* PPT *algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

Proof: We show how to construct a collision for the vectorial Zémor-Tillich with key $P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$ from a collision for ZesT with the same parameters and vice-versa; the result then follows from Proposition 5.11. Let (m, m') be a collision on ZesT: we have $m \neq m'$ and

$$\begin{aligned} &H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), (m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right))\right) \\ &\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), (m'||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right))\right). \end{aligned}$$

The messages $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ and $m'||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right)$ collide for the vectorial version and are distinct. On the other hand, it is clear that any collision on the vectorial version is also a collision on ZesT. □

The equivalence result of Proposition 9.1 is nearly tight. On one side, a solution to the balance problem immediately gives a collision on ZesT. On the other side, $\log_2 n$ bits of security are "lost" from the vectorial to the matrix version of Zémor-Tillich in the proof of Proposition 5.11. The collision resistance of ZesT is not optimal as its output has $2n$ bits while the collision attacks of Section 5.4 will find collisions for the vectorial version of Zémor-Tillich in time $2^{n/2}$. In Section 9.5.3, we will suggest a modification of ZesT that reaches optimal collision resistance.

9.2.2 Preimage resistance up to the collision resistance level

The preimage resistance of ZesT follows from the hardness of the balance problem corresponding to the Zémor-Tillich hash function. We give here a proof that provides a preimage resistance guarantee but only up the $n/2$ bits of the collision resistance level.

Proposition 9.2 *If there exists a* PPT *algorithm that breaks the preimage resistance of* ZesT*, then there exists a* PPT *algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

Proof: The result is immediate as ZesT processes arbitrary-length bit sequences and it is collision resistant if the balance problem is hard (see [232], Section 2.2.4 and Proposition 9.1). The informal argument is as follows. Suppose there exists an efficient algorithm A computing preimages, then there exists an efficient algorithm B that finds collisions: B chooses a random message m, computes its hash value, gives the hash to A and receives m' from A. As each hash value has a lot of preimages on average, the messages m and m' are likely to be different hence to form a collision. □

The result is not tight as there does not currently exists any algorithm able to compute preimages for ZesT in time $2^{n/2}$. Indeed, we argue in Section 9.3 that the actual preimage resistance level of ZesT seems closer to $2n$ bits.

9.2.3 Second preimage resistance up to the collision resistance level

The second preimage resistance of ZesT also follows from the hardness of the balance problem corresponding to the Zémor-Tillich hash function.

Proposition 9.3 *If there exists a* PPT *algorithm that breaks the second preimage resistance of* ZesT*, then there exists a* PPT *algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

Proof: Identical to the proof of Proposition 9.2. □

This result is tight as there exists an algorithm computing second preimages in time $2^{n/2}$. Indeed, given a message m, there exists an algorithm computing a preimage of $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) := H_{ZT}^{vec}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)$ in time $2^{n/2}$: this algorithm first computes a matrix $M \in SL(2, \mathbb{F}_{2^n})$ such that $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) M = \left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$

and it then applies the preimage algorithm of Section 5.4 to M. To compute a second preimage of ZesT, it suffices to give $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ to this algorithm; as ZesT processes arbitrary-length inputs the message m' returned is likely to be different from m. Moreover,

$$\begin{aligned}
&H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), (m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right))\right)\\
&\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)\right)\\
&\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right)\right)\\
&\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), (m'||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m'\right))\right).
\end{aligned}$$

9.3 Other security aspects of ZesT

ZesT is Zémor-Tillich with Enhanced Security inside. In this section, we give security properties of ZesT that cannot be proved based on the hardness of the balance problem corresponding to Zémor-Tillich, but that still appear very likely.

9.3.1 Output distribution

We argue that for long messages, the output distribution of ZesT is close to uniform. Given a message m, ZesT outputs the value

$$\begin{aligned}
&H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), (m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right))\right)\\
&\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)\right)
\end{aligned}$$

which can be seen as the result of two consecutive walks determined by the bits of m and of $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) := H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$. Although no good bound is known on the eigenvalues of $\mathcal{ZT}$, convergence of random walks is guaranteed in these graphs hence also in $\mathcal{ZT}^{vec}$ graphs (see Sections 5.2 and 5.5.3). For long messages, the uniform distribution of $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ follows from the expanding properties of $\mathcal{ZT}$ and $\mathcal{ZT}^{vec}$ graphs. The second walk of $2n$ bits performed from $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$ should not affect this distribution because the bits of $H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ are expected to be reasonably random and independent of those of m. Under this independence assumption, the output distribution of ZesT on long messages is provably close to the uniform distribution.

9.3.2 Preimage resistance

The preimage resistance of ZesT is much better than the $n/2$ bit security that can be proved based on the hardness of the representation problem.

As preimages of the first round can be computed in time $2^{n/2}$, we may try to fix the value h' after this first round and to recover the message with this additional constraint. However, finding an h' value that may satisfy the equation $h = h'H_{ZT}\left(P_n(X), h'\right)$ when h is given seems to be a hard problem, that cannot be solved faster than by exhaustive search methods.

A preimage on ZesT implies a preimage on H_{ZT}^{vec} in the second round that has the form $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$. The preimage algorithm of Section 5.4.3 can be easily modified to compute some kinds of particular preimages on the vectorial version. For example, there exists an algorithm finding preimages that start and end with some given constant bitstrings. However, computing preimages of the form $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ faster than by exhaustive search seems to be out of reach.

The preimage attacks of Section 5.4.3 cannot be extended to messages of the form $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$. For generic messages, we could concatenate various messages colliding for the projective version into a collision for the vectorial version. The approach does not work here because the concatenation of two messages of the form $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ is not a message of the form $m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$ in general.

For generic messages, we could also follow a "meet-in-the-middle" strategy to get preimages at the price of collisions. In the second round of ZesT, this approach is no longer possible because of the redundancy between the left-most and the right-most bits of the message that is given to the second round. More generally, it seems impossible to exploit the mathematical structure of ZesT (in particular the associativity of the matrix product) to improve generic preimage attacks against the second round because of the redundancy introduced between the bits of m and those of $H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right)$.

The actual preimage resistance of ZesT is therefore of $2n$ bits, hence much better than the $n/2$ bits security obtained from the hardness of the balance problem. For applications that only require 60 bits of preimage resistance (without requiring collision resistance), parameters as small as $n = 31$ will therefore be safe.

9.3.3 Issues in Zémor-Tillich that are removed in ZesT

ZesT does not present any apparent malleability property nor any apparent predictable behavior contradicting the intuition of pseudorandomness.

Unlike Zémor-Tillich and its vectorial variant, ZesT cannot be inverted on short messages. In these functions, the invertibility comes from the absence of modular reductions when the message size is only slightly larger than n. The issue is removed in ZEST because at least $2n$ bits are hashed in the second round.

ZEST is not malleable. As an example, let us consider a simple malleability issue of H_{ZT}^{vec} that is a relation between the hash values of m and $m' = m||0$:

$$\text{if } H_{ZT}^{vec}(m) = h_1||h_2 \text{ then } H_{ZT}^{vec}(m') = (h_1X + h_2)||h_1.$$

We point out that the malleability of the vectorial Zémor-Tillich is limited to addition and/or suppression of bits on the right side of unknown messages. In particular, it is not possible to modify a hash value according to a change in the middle bits of an unknown message.

Let us now consider $\text{ZEST}(m) = H_{ZT}^{vec}(m||H_{ZT}^{vec}(m))$. Although they are strongly correlated, the hash values $H_{ZT}^{vec}(m)$ and $H_{ZT}^{vec}(m')$ differ in many middle bits in general, so $\text{ZEST}(m)$ and $\text{ZEST}(m')$ are completely uncorrelated. Of course, there exist some particular values (m, m') such that $H_{ZT}^{vec}(m)$ and $H_{ZT}^{vec}(m')$ are very close, for example differ by only the last bit, but finding such a pair without inverting H_{ZT}^{vec} already seems a hard problem. Moreover, any such m and m' that we could find would differ in many bits, so again $\text{ZEST}(m)$ and $\text{ZEST}(m')$ would be completely uncorrelated.

In Section 9.5.2 below, we provide further evidence that ZEST has no apparent weakness, based on analysis carried out with the pseudorandom tests of the Dieharder [5].

9.3.4 Security as a MAC

ZEST can be used as a message authentication code $\mathcal{M}\text{ZEST} = (Gen, Mac, Ver)$. The Gen and Mac algorithms of $\mathcal{M}$ZEST are just the key generation and the hash algorithms of ZEST. Of course, the key remains secret here, and it is important that both elements $P_n(X)$ and $\begin{pmatrix} a_0 & b_0 \end{pmatrix}$ remain secret. On input (s, m, t), the verification algorithm simply checks whether $t = \text{ZEST}(s, m)$.

$\mathcal{M}$ZEST is essentially HMAC used with the vectorial version of the Zémor-Tillich hash function. Although the weaknesses of this last function are not present in the functions usually employed with HMAC, we argue that $\mathcal{M}$ZEST is a secure MAC algorithm.

Key recovery against $\mathcal{M}$ZEST seems to be a hard problem. A ZEST key is made of two components, an irreducible polynomial $P_n(X)$ and an initial vector $\begin{pmatrix} a_0 & b_0 \end{pmatrix}$. We argue that recovering the whole key has a cost 2^{2n} even if the polynomial $P_n(X)$ can be recovered in time 2^n.

Let us first suppose that the polynomial $P_n(X)$ is not known to the adversary. If the adversary knew the initial vector $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$, he could easily recover $P_n(X)$ as follows. The adversary would send the void message to the *Mac* algorithm and receive an answer which equals $H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)\right)$. The adversary could also compute the hash value by itself without performing the reductions and subtract the hash value returned by the *Mac* algorithm to obtain a vector $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) \in \mathbb{F}_2[X]$. The polynomial $P_n(X)$ would be an irreducible factor of $\gcd(a, b)$ with degree n. If needed, the adversary would make an additional query to the *Mac* algorithm to discriminate between alternative possible factors of degree n.

If the adversary does not know the initial vector, he can still recover the polynomial $P_n(X)$ with 2^n $\mathcal{M}$ZesT queries as follows. After 2^n queries, the adversary is likely to find two messages with the same MAC value. With a probability of about 50%, these two messages provide a collision (m, m') on the vectorial version of Zémor-Tillich. The adversary then computes the Zémor-Tillich hash values M, M' of m and m' without performing the modular reductions. The polynomial $P_n(X)$ is an irreducible factor of degree n of $\det(M + M')$. As this determinant may have more than one polynomial factor of degree n and as the adversary does not know whether the collision he obtained for the MAC was a collision for the first round, he needs a few MAC collisions to identify the right polynomial.

Let us now suppose that the polynomial $P_n(X)$ is known to the adversary who wants to recover the initial vector. From messages m_i of his choice and the corresponding hash values $h_i :=$ ZesT$(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m_i)$, the adversary tries to recover $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$. This is equivalent to finding any $\left(\begin{smallmatrix} a_i & b_i \end{smallmatrix}\right)$ such that $\left(\begin{smallmatrix} a_i & b_i \end{smallmatrix}\right) = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m_i\right)$ because $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} a_i & b_i \end{smallmatrix}\right)\left(H_{ZT}\left(P_n(X), m_i\right)\right)^{-1}$. The task of the adversary now consists in solving the equation

$$\left(\begin{smallmatrix} a_i & b_i \end{smallmatrix}\right) H_{ZT}\left(P_n(X), \left(\begin{smallmatrix} a_i & b_i \end{smallmatrix}\right)\right) = h_i$$

for one of the h_i. Solving this equation seems hard because of the redundancy in the unknown; the preimage attack of Section 5.4.3 does not extend to this case. We believe that when the polynomial is known, the adversary cannot recover the initial vector faster than in time 2^{2n}.

Message-extension attacks against $\mathcal{M}$ZesT are defeated by the second round of ZesT. These attacks are possible for any iterative hash function, in particular all Merkle-Damgård-based hash function and all expander hashes. The attack is prevented in ZesT in exactly the same way as in the HMAC construction: the second round destroys the block structure and prevents the adversary from having access to the result of an iterative hash function.

Forging $\mathcal{M}$ZesT seems to require 2^n MAC queries plus a computation time $2^{n/2}$. As soon as the polynomial $P_n(X)$ is known by the adversary, a forgery for this MAC is feasible in time $2^{n/2}$: the forger can compute a collision (m, m') for the Zémor-Tillich hash function, query the Mac algorithm on m to receive t, and return (m', t) as a valid forgery. When the polynomial is not known, the adversary cannot compute collisions for the Zémor-Tillich hash function. Moreover, its oracle access to $\mathcal{M}$ZesT does not help it to attack the Zémor-Tillich hash function as he only accesses the output of the second round. The malleability of the first round is not useful either to the adversary for the same reason. We have found no forgery algorithm faster than our best partial key recovery algorithm on $P_n(X)$ followed by a collision attack on the Zémor-Tillich hash function.

The trapdoor attacks and weak keys issues present in ZesT do also affect $\mathcal{M}$ZesT. However, when the key generation algorithm is not trusted, the techniques sketched out in Section 9.1.3 to protect ZesT will also protect $\mathcal{M}$ZesT.

9.3.5 Connections with HMAC and other iterative designs

The design of ZesT is inspired by HMAC and by traditional block cipher and compression function designs: the mathematical structure remaining after the first round of ZesT is destroyed in its second round. However, most existing security results on HMAC assume hypotheses on the hash function that are clearly not satisfied by the vectorial Zémor-Tillich hash function, and block ciphers and compression functions usually have much more than just two rounds. The collision resistance of ZesT is guaranteed with a single round; a second round is necessary to obtain "extra" pseudorandom properties; the second round is also sufficient because the round function is already very strong.

ZesT looks very similar to NMAC and HMAC (Section 2.6.1). The collision resistance transfers from the vectorial Zémor-Tillich hash function to ZesT, exactly like in these constructions. On the other hand, existing security results on the pseudorandom and MAC security of HMAC and NMAC cannot be used for ZesT. NMAC and HMAC were built for iterative hash functions whose compression functions have no apparent weaknesses. In contrast, the vectorial version of Zémor-Tillich is highly malleable and any "compression function" we could define from it by fixing a block size would also be malleable. In particular, this compression function would definitely not be pseudorandom nor a secure fixed-length MAC. Unlike its collision resistance, the pseudorandomness of ZesT follows from the iteration and not from the single rounds.

The iterative design of ZesT also appears in many block ciphers and in compression functions, typically with 16 to 64 rounds. In contrast, ZesT only has 2 rounds. Block ciphers or traditional hash functions would become invertible if their round number was decreased. In contrast, a single round of ZesT is already preimage and collision resistant because it is a whole hash function and it is therefore much stronger than the simple components used in block ciphers and compression functions.

9.4 Efficiency of ZesT

ZesT provides great recipes for many diets: it is really practical in a wide range of applications. We describe software implementations in Section 9.4.1, FPGA implementations in Section 9.4.2 and lightweight implementations in Section 9.4.3. Finally, we show in Section 9.4.4 how to exploit in ZesT the inherent parallelism of the Zémor-Tillich hash function.

For FPGA and lightweight implementations, we will base our performance estimations on the implementations in [88] of the function introduced in [208]. This function, that we will call ZT' in this thesis, is very similar to ZesT. The only difference is the introduction of an XOR by a constant between the first and the second round:

$$\begin{aligned} &ZT'\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) \\ &\quad := H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), \left(m||H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) \oplus c\right)\right). \end{aligned} \tag{9.1}$$

The constant c is equal to the binary representation of pi in [208].

9.4.1 Efficiency of ZesT in software

ZesT recursively uses a very simple operation on a state $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right)$. Depending on the next message bit, the state is updated to $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) A_0 = \left(\begin{smallmatrix} aX+b & a \end{smallmatrix}\right)$ or to $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) A_1 = \left(\begin{smallmatrix} aX+b & aX+b+a \end{smallmatrix}\right) = \left(\begin{smallmatrix} aX+b & (aX+b)+a \end{smallmatrix}\right)$. After processing all the message bits, the result is seen as a bitstring and processed in turn. Messages of μ bits therefore require to process $\mu + 2n$ bits for the first and the second rounds together.

The arithmetic is in a field of characteristic 2 and is thus very efficient. In our C implementation on a 32-bit architecture, we represent an element $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + ...a_1X + a_0$ as an array A of $L := \lceil \frac{n}{32} \rceil$ integers ($L := \lceil \frac{n}{64} \rceil$ in 64-bit architectures). An addition requires only L XORs, and a multiplication a by X requires L SHIFTs by one bit and one polynomial modular reduction. The operations $aX + b$ and $aX + a$ can be performed with L SXORs and a modular reduction. The polynomial reduction in the computation of $aX + b$ can be done by testing the left-most bit of a: if this bit is equal to 1, we need L XORs operations of the bits of $aX + b$ with the bits of $P_n(X)$. For long messages, the TEST instruction will return 1 half of the times.

Let t_0 and t_1 be the average times needed respectively to process a bit 0 and 1, let t_{XOR} and t_{SXOR} be the times needed to perform a word XOR and SXOR, and let t_{TEST} be the time needed to perform a TEST instruction. According to our analysis, t_0 and t_1 are respectively

$$\begin{aligned} t_0 &= Lt_{SXOR} + \frac{L}{2}t_{XOR} + t_{TEST} + C_0, \\ t_1 &= Lt_{SXOR} + \frac{3L}{2}t_{XOR} + t_{TEST} + C_1. \end{aligned}$$

where C_0 and C_1 are constant overhead times. If we neglect the TEST instruction and the overhead times and if we approximate $t_{SXOR} \approx t_{XOR}$, processing one bit requires on average $2Lt_{XOR}$ instructions. The total time needed to evaluate the ZEST hash value of a message of length μ is therefore

$$2(\mu + 2n)Lt_{XOR}.$$

For long messages, this time is essentially proportional to the message length and inversely proportional to the architecture size.

ZEST can be cut into small pieces; it is very scalable to any granularity. Implementing ZEST on an 8-bit processor or on the other hand on a graphical accelerator with a 512-bit data bus is just as easy as implementing ZEST on standard 32 or 64-bit processors, and the implementation speed will be directly proportional to the architecture size. If the architecture is larger than n, ZEST only requires two XORs per message bit, plus $4n$ XORs for the second round.

ZEST algorithm was implemented in C to get running time estimations for various parameters sizes. All tests were performed on a 64-bit Intel Xeon E5420 2.5GHz 16Go DDR2 Ram. The OS was Debian using 32-bit kernel 2.6.26. Test vectors for performance evaluation were 500Mo random files generated using `/dev/urandom`. The values chosen for the parameter n were the smallest primes smaller than 32, 64, 128, 160, 224, 256, 384, 512 and 1024, and for each n a random polynomial of degree n was selected for $P_n(X)$.

Performance results are presented in Table 9.1. The results differ significantly from the above analysis. First, the analysis seems to become valid only for large values of the parameter n. This may be due to various overheads, in particular to *for* loops present in the code for scalability reasons that should better be unrolled for short parameters. Second, we observe that ZEsT-61 and ZEsT-251 are respectively more efficient than ZEsT-31 and ZEsT-157: this might be due to the subjacent 64-bit architecture of our evaluation platform and to scalability options taken in our code.

In Table 9.2, the performances of ZEsT are compared to the SHA algorithm evaluated with the `sha1sum`, `sha256sum` and `sha512sum` functions of the linux kernel. At comparable collision resistances, ZEsT-127, ZEsT-251 and ZEsT-509 are respectively 10, 10 and 4 times less efficient than SHA-1, SHA-256 and SHA-512. At comparable preimage resistances (which is $2n$ bits for ZEsT), ZEsT-127 and ZEsT-251 are respectively 7 and 2 times less efficient than SHA-256 and SHA-512. If we only rely on the $n/2$ "provable" bits of preimage resistance, then at comparable preimage resistances, ZEsT-509 and ZEsT-1021 are respectively 17 and 7 times less efficient than SHA-256 and SHA-512.

Table 9.1: Estimated running time (seconds) of ZEsT on a 500Mo file with parameters of various sizes on a 2.5GHz 32-bit architecture, and corresponding cycles/byte

ZEsT	Time (s)	(cycles/byte)
ZEsT-31	40.3	210
ZEsT-61	31.3	163
ZEsT-127	40.3	210
ZEsT-157	61.9	322
ZEsT-223	72.5	377
ZEsT-251	60.1	313
ZEsT-383	77.9	405
ZEsT-509	97.2	505
ZEsT-1021	173.5	902
ZEsT-2039	331.7	1725

Table 9.2: Comparison of SHA and ZEsT at the same collision resistance levels

CR	SHA	(cycles/byte)	ZEsT	(cycles/byte)
$\approx 2^{64}$	SHA-1	19	ZEsT-127	210
$\approx 2^{128}$	SHA-256	30	ZEsT-251	313
$\approx 2^{256}$	SHA-512	136	ZEsT-509	505

The basic algorithm described above can be improved by grouping the computation of consecutive message bits. As an example, let us consider the process of two consecutive bits: depending on these two bits, the state (a, b) is updated to

$$\begin{aligned}
(a,b)A_0A_0 &= (aX^2+bX+a, aX+b),\\
(a,b)A_0A_1 &= (aX^2+bX+a, aX^2+aX+bX+a+b),\\
(a,b)A_1A_0 &= (aX^2+aX+bX+a+b, aX+b),\\
(a,b)A_1A_1 &= (aX^2+aX+bX+a+b, aX^2+bX+a).
\end{aligned}$$

The last product can be computed by computing first $aX+b$, then $(aX+b)X+a = aX^2+bX+a$ and finally $(aX^2+bX+a)+(aX+b)$. The cost of this sequence of instructions is 2 XORs and 1 SXOR (plus the polynomial reductions). The trivial "one bit at once" implementation computes $aX+b$ and $(aX+b)+a$ then $(aX+b)X+(aX+b+a)$ and $((aX+b)X+(aX+b+a))+(aX+b)$ hence it requires an additional SXOR. This simple observation leads to a speedup of 25% for one fourth of the 2-bit sequences.

The idea of matrix grouping is easily generalized to larger bit sequences with the help of code-generation programs. More specifically, a program can be written that

- Computes the vector-by-matrices product;
- Looks for the best data paths with respect to the operations $a \to aX$, $(a,b) \to aX+b$ and $(a,b) \to a+b$;
- Selects the very best data path according to an optimization function including the computation times of individual operations and the number of registers needed in a C implementation;
- Writes an optimized C code computing the products.

The ZEST algorithm is so simple that code-generation programs may be easily tuned to any computer architecture and may also include more elaborate grouping strategies, for example based on Huffman coding.

Finally, we point out that the performances of ZEST will be greatly improved by using the multimedia sets of instructions available on modern processors.

9.4.2 FPGA implementation

In this section, we describe high-speed implementations of ZEST on FPGA. These implementations are of interest in applications where several messages are to be hashed and a high throughput is required, such as for virtual network servers. The throughput per area metric makes sense here since the goal is to jointly minimize the execution time and the area: if the throughput of the resulting implementation is too low, any superior throughput can be reached by simply gathering several identical circuits.

The main feature of ZesT is that it considers the bits of the message one after the other. The dependency between the intermediary results of the hash function suggests an iterative architecture where the message is processed one bit at the time. The efficiency can be improved by processing s consecutive bits at each step, i.e., by partially unrolling the main loop. The optimal value for s (the unroll factor) is determined empirically. When processing one bit of the message, the bits of the two entries a and b of the matrix can be efficiently computed in parallel. Indeed, the operations involved in the hash function are simple: they essentially consist in XOR operations between two or three operands and 1-bit left shifts. These bitwise operations allow computing the bits of a and b in parallel without decreasing the frequency, which would not be the case it there was a carry propagation for instance. The parallel approach is better than the serial one: the latter alternative would involve a wordwise processing and would therefore require extra control logic to select and route the appropriate words during the execution of the function, resulting in a lower throughput per slice efficiency.

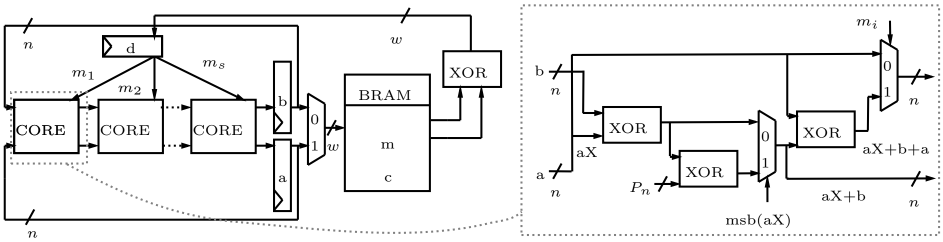

Figure 9.1: FPGA architecture for ZT' proposed in [88]

Figure 9.1 presents the architecture proposed in [88] for ZT' which is essentially ZesT with an additional XOR between the first and the second round (see Equation 9.1). It is made of a central core processing one bit of message, and of storage elements. The core can be replicated s times in order to process s consecutive message bits. Each of these bits, m_i with $0 < i < s - 1$, comes from a w-bit shift register denoted d containing a word of the message to hash. The partial results are stored in two n-bit registers, denoted a and b, and are used as inputs when processing the next s bits of the message. They loop in the circuit until the end of the message is reached, then they are loaded in the block ram (BRAM) and are reused as if they were the continuation of the message in the final phase.

The BRAM also stores the message, denoted m, and the constant c (pi binary representation) by words of $w = 32$ bits. The two output ports of the BRAM are connected to a w-bit XOR that is used to implement the function σ, i.e., the XOR operation with c preceding the final phase. Before the final phase, zero is outputted in place of c to correctly route the words of m to the core of the circuit through the w-bit XOR gate.

The circuit described in Figure 9.1 processes s bits of the message per clock cycle. The throughput achieved can be approximated by the product of the frequency and s if the processing of the final phase is negligible. This is valid for sufficiently long messages with respect to the $2n$ additional bits of the final phase. In the following of this section, we assume that the messages fulfill this condition.

The parameter s must be properly chosen in order to maximize the throughput per slice. A small s ensures a large operation frequency together with a small occupied area. Increasing s is interesting in the sense of the considered metric, since it is proportional to the throughput of the circuit while being proportional only to the core (as defined in Figure 9.1) which is only a fraction of the required area. However, increasing s adds logic operators to the longest path, resulting in a lower maximum operating frequency. The optimal value of s was therefore determined by an empiric study (Table 9.3).

Table 9.3: Implementation results of ZT'-127 in function of the s, the number of message bits processed per clock cycle

s parameter	Area [Slices]	Frequ. [MHz]	Through. [Mbits]	Through./Area [Mbits / Slice]
1	262	220	220	0.84
2	377	215	430	1.14
3	515	185	555	1.08
4	596	170	680	1.14
5	597	160	800	1.34
6	647	130	780	1.21
7	794	120	840	1.06
8	901	110	880	0.98

The function ZT' was implemented on the Xilinx Virtex-2 XC2V2000-6.

Synthesis and place and route were performed with ISE 8.2 and testing and debugging with Modelsim SE 6.1. The first implementation results determined the optimal value for s, the number of message bits processed per clock cycle: the highest throughput per slice was obtained with $s = 5$ for $n = 127$, 251 and 509. Implementation results for these parameters are given in Table 9.4. The impact of n on the frequency is moderate as increasing n does not add logic operators to the longest path (as s does). The small frequency drop is likely due to larger routing delays. On the other hand, the area may be approximated by a linear function of n:

$$\text{Area} = 3.3n + 200.$$

The area is nearly proportional to n as the only constant parts of the circuits are the control logics and the BRAM.

Table 9.4: Implementation results for ZT' with $s = 5$

n	Area [Slices]	Frequency [MHz]	Throughput [Mbits]	Throughput/Area [Mbits / Slice]
127	597	160	800	1.34
251	1044	140	700	0.67
509	1850	135	675	0.37

The implementation results for ZT' provide fair estimates of the performances of ZesT. Due to the absence of the constant c in ZesT, the usage of BRAM and of control logics will slightly decrease. The frequency should be on the same order as the longest data path remains unchanged. FPGA implementations of ZesT will therefore have slightly better but comparable throughput/area ratio as ZT'.

In Table 9.5, we compare these results to the very optimized implementations of SHA proposed in [73] and [72] representing the current state-of-the-art in terms of achieved throughput per occupied area. The results show that the performances of ZT' and of SHA in terms of throughput per slice are in the same order. At comparable level of collision resistance, our implementations of ZT'-127, ZT'-251 and ZT'-509 are about twice less efficient than the state-of-the-art implementations of SHA-1, SHA-256 and SHA-512 respectively.

These implementations already reach the level of performances of SHA, but we believe that they could be further improved by introducing pipeline

Table 9.5: Comparison of the performances of the high-speed implementations of ZT' with SHA.

	Collision Resistance	Area [Slices]	Frequ. [MHz]	Through. [Mbps]	Through./Area [Mbps / Slice]
SHA-1 [73]	2^{63}	533	230	1435	2.7
ZT'-127	2^{64}	597	160	800	1.34
SHA-256 [72]	2^{128}	797	150	1184	1.49
ZT'-251	2^{126}	1044	140	700	0.67
SHA-512 [72]	2^{256}	1666	121	1534	0.92
ZT'-509	2^{255}	1850	135	675	0.36

stages between the s cores. This technique should allow increasing the operating frequency while still processing s bits per clock cycle, these bits being from different messages this time. This would significantly increase the throughput at a relatively small area cost, i.e., mainly a $2n$-bit register per pipeline stage. For instance, only one pipeline stage could in theory increase the frequency of the design of ZT'-127 with $s = 8$ (110MHz, see Table 9.3) to the frequency of the design with $s = 4$ (170 MHz, see Table 9.3), resulting in a throughput of roughly 1300 Mbps in place of 880, i.e., roughly 50% improvement.

Since ZEST does not process the message by fixed size blocks, the control part of the pipelined architecture will be more complicated in the case of messages of different sizes. Indeed, when one message in the pipeline will come to the processing of the final phase or when it will be fully hashed, the computations for all messages will be irregularly interrupted. In applications where the message sizes are integer multiples of some block size, the control of the pipelined architecture is simplified as the interruptions for the final phase and the final result happen at the same processing steps for each block.

9.4.3 Lightweight implementations

We now study the performances of ZEST in constrained environments such as for RFID tags authentication. Like in the previous section, we give estimations based on the lightweight implementations of ZT' that we proposed in [88]. As shown in Table 9.3, ZT'-127 with $s = 1$ already occupies a relatively small area for the architecture presented in Section 9.4.2. We now modify this architecture to focus on area reduction.

Figure 9.2 presents the architecture of our lightweight implementation of

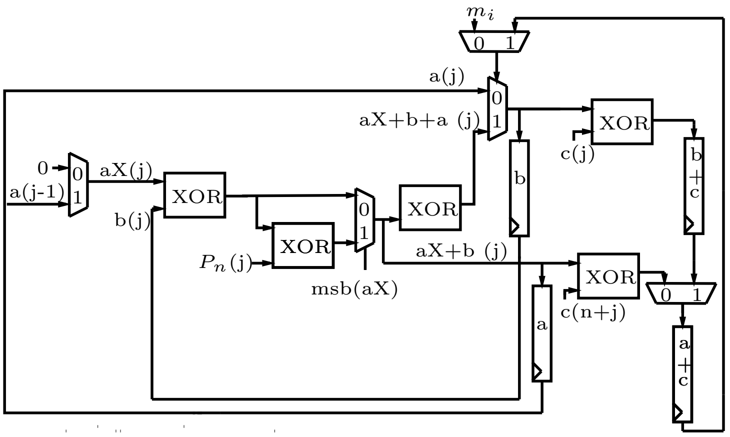

Figure 9.2: Lightweight architecture for ZT' proposed in [88]

ZT'. The first main change introduced consists in computing the entries a and b one bit at the time instead of all bits in parallel in order to save area by replacing n-bit gates by 1-bit ones. This is illustrated on the figure with the presence of the j suffix, with $0 \leq j \leq n-1$, that indicates that the circuit operates on one-bit signals in order to compute one bit of a and b per clock cycle. Processing one message bit m_i therefore requires n clock cycles instead of one previously.

The second main change involves the storage elements. In lightweight implementations, large blocks of memory like BRAM are no longer available and are therefore replaced by two registers (labeled $a+c$ and $b+c$) that store the result of the XOR operation between the intermediary result and the constant c, which is hardcoded. The outputs of these registers are used as the continuation of the message during the final phase. This architecture assumes that the message is loaded in the circuit one bit at the time and therefore does not require a dedicated storage element. The four n-bit registers in this architecture are shift registers that do not have a parallel load since the circuit operates on one-bit signals. Their 1-bit inputs and outputs are represented as wires respectively at their tops and bottoms. At each clock cycle, the register a outputs two bits $a(j-1)$ and $a(j)$, the register b gives $b(j)$ and both registers take as inputs the two bits computed by the circuit. The registers $a+c$ and $b+c$ also input these bits but only after a XOR operation with the constant bits $c(n+j)$ and $c(j)$. During the final phase, the output of the register $b+c$ is redirected toward the register $a+c$, which provides the bits of the continuation of the message. At the end of the final phase, the hash values, stored in the registers a and b, are also retrieved one bit at the time.

As suggested earlier, the latency of this circuit when hashing a message m containing μ bits is $(\mu + 2n).n$ clock cycles. The corresponding throughput cannot be approximated any more by leaving aside the processing of the additional $2n$ bits of the final phase since the messages are likely to be small in the context of lightweight implementations. In the following, it is computed based on the formula of the latency given above with $\mu = 512$, i.e., assuming consecutive messages of 512 bits. This size is arbitrary as there is no fixed block size in the ZT' function.

Unless n is very small, the registers occupy a significant part of the total area required. In a first approximation, the area needed is made of a constant part dedicated to control and computing logics and of a second part proportional to n for the four registers. As the computation part is small, the main part of the area is used by the registers unless n is very small.

Compared to ZT', ZEST will occupy significantly less area and will use a comparable frequency. Removing the constant between the two rounds allows sparing two out of the four registers of Figure 9.2. This gain becomes more significant when the parameter size increases because the area taken by the registers also becomes more significant. The frequency of ZEST is not changed compared to ZT' because the longest data path is in the processing of message bits which is unchanged.

The design for ZT' was synthesized using Synopsys Design Analyzer version Y-2006.06 with the CMOS65 library of STMicroelectronics. To provide a comparison with the architecture of the preceding section, it was also implemented on FPGA. For n=127, it requires only 73 slices with a frequency of 145MHz. This particularly small area requirement is due to the use of compact SRL16 registers to implement the shift registers (with no parallel load), which significantly diminishes the number of slices used.

Table 9.6 summarizes the results concerning ZT' and other hash functions based on the comparison performed in [49]. Results concerning the AES block cipher are also given as a reference. The results for SQUASH are based on the estimate performed in [123], which describes a lightweight implementation on the Xilinx Virtex-4 LX FPGA. This estimate must be seen as an upper bound. Finally, we also synthesized the ZT'-127 and ZT'-251 FPGA designs with $s = 1$ to evaluate the area reduction obtained by the lightweight design.

ZEST will be very efficient in terms of occupied area with respect to current hash functions. Table 9.6 shows that both the lightweight and high-speed (with $s = 1$) versions of ZT'-127 already outperform the hash functions SHA-1 and MD5. Lightweight ZT'-127 is a little smaller than the state

Table 9.6: Comparison of the performances of the lightweight implementation of ZT' with other hash functions and the AES block cipher

	Output size	Through. at 100kHz [kbps]	Through /Area [bps/GE]	Logic process	Area [GE]
MD5 [99]	128	83.7	10	0.13μm	8400
SHA-1 [99]	160	40.2	4.9	0.35μm	8120
SHA-256 [99]	256	45.4	4.2	0.35μm	10868
SQUASH [123]	32	< 0.1	< 0.02	estimate	<6000
AES-128 [98]	128	12.4	3.7	0.35μm	3400
DM-PRESENT-80 [49]	64	14.6	9.1	0.18μm	1600
H-PRESENT-128 [49]	128	11.5	4.9	0.18μm	2330
ZT'-127 (lightweight)	254	0.52	0.18	65nm	2945
ZT'-251 (lightweight)	502	0.20	0.04	65nm	5517
ZT'-127 (s = 1)	254	66.7	17.8	65nm	3752
ZT'-251 (s = 1)	502	66.7	9.2	65nm	7267

of the art implementation of the AES block cipher proposed in [98]. The area requirements and collision resistances of ZT' and SHA are compared in Table 9.7, illustrating the inferior area costs for ZT' at a comparable collision resistance. ZT'-127 requires roughly one third of the area of SHA-1 while ZT'-251 needs half of the area of SHA-256. ZT'-127 is a little less compact than H-PRESENT-128, the hash function recently proposed in [49] based on the block cipher PRESENT.

ZEST-127 is comparable to DM-PRESENT-80 and it outperforms even H-PRESENT-128 for the same collision resistance. Based on our results for n=127 and n=251, the area required for the lightweight ZT' may be approximated by the function Area = $20n + 300$. The area for ZEST can therefore be roughly approximated by Area = $10n$+300 as half of the registers are removed. This leads to approximations of 1600 and 2900 gates equivalents for ZEST-127 and ZEST-251 respectively.

For some applications, collision resistance is not required and a moderate level of security is sufficient (60-bit or 80-bit security) [243]: for example, many RFID protocols only rely on preimage resistance. ZEST turns out to be a very interesting candidate for these applications. As explained in Section 9.3.2, the preimage resistance of ZEST-n is at least $2^{n/2}$ based on current knowledge on the balance problem, but it actually seems to be 2^{2n}.

In Table 9.8, our lightweight implementation of ZT'-127 is compared to SQUASH and DM-PRESENT-80 in terms of preimage resistance and lightweight implementations. ZT'-127 is twice as small as SQUASH. The function DM-PRESENT-80 [49] is nearly twice as small as ZT'-127 but (according to our estimations above) comparable to ZEST-127. If we only rely

Table 9.7: Comparison of the collision resistance and area cost of SHA with the lightweight implementation of ZT' and the approximation for ZesT.

	Collision Resistance	Area [GE] (rel.)
SHA-1 [98]	2^{63}	8120 (1)
ZT'-127 (lightweight)	2^{64}	2945 (0.36)
ZesT-127 (approximation)	2^{64}	1600 (0.20)
SHA-256 [98]	2^{128}	10868 (1)
ZT'-251 (lightweight)	2^{126}	5517 (0.51)
ZesT-251 (approximation)	2^{126}	2900 (0.27)

on the "provable" preimage resistance of ZesT, lightweight implementations of ZesT-127 are therefore comparable to those of DM-PRESENT-80. However, based on the "heuristic" preimage resistance argued in Section 9.3.2, DM-PRESENT-80 should be compared with a version of ZesT four times smaller (for example, ZesT-31). According to our estimations for ZesT-127 and ZesT-251, ZesT-31 will probably require less than 1000GE, beating by far even DM-PRESENT-80.

Table 9.8: Comparison of the preimage resistances and area costs of lightweight implementations of ZT' and other one-way hash functions.

	Preimage Resistance	Area [GE] (rel.)
SQUASH [123]	2^{32}	<6000 (1)
DM-PRESENT-80 [49]	2^{64}	1600 (0.27)
ZT'-127 (lightweight)	2^{64} - 2^{256}	2945 (0.49)

As pointed out above, ZesT already occupies a small area if the high-speed design of Section 9.4.2 is used with $s = 1$. In practice, this implementation will probably be more suitable for area-constrained applications than the lightweight version presented in this section. As our design choice here was to minimize the area, our implementation has a low throughput resulting in a long latency and an important energy consumption. However, the flexibility of the ZesT function allows to raise the throughput easily by increasing the number of bits of a and b processed in parallel at the cost of little additional logic. The two extreme points of this tradeoff between area and throughput are our first implementation with $s = 1$ and our lightweight implementation; the first one has a throughput 128 times as high for only 30% more area. The optimal point in practice will probably be closer to the the first one but the results of this section may be understood as a lower bound for area. Wherever the tradeoff is set, ZEST is a very interesting hash function in the context of lightweight applications.

9.4.4 Exploiting parallelism

ZEST can be cut into small pieces (see Figure 9.3). It is particularly well-suited for parallelism in the message computation. We point out that unlike many hash functions recently proposed, ZEST has a serial and a parallel modes that describe exactly the same function. Indeed, let us suppose that we have N computing units for computing the ZEST hash value of a long message.

Figure 9.3: Citrus' ZEST in serial and parallel modes

For any $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \in \mathbb{F}_{2^n} \setminus \{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)\}$ and for any bitstrings $m_1, m_2, ..., m_{N'} \in \{0,1\}^*$, we have

$$\begin{aligned} &H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m_1||m_2||...||m_{N'}\right) \\ &\qquad = H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m_1\right) H_{ZT}\left(P_n(X), m_2\right) ... H_{ZT}\left(P_n(X), m_{N'}\right). \end{aligned}$$

Moreover, the matrix version of Zémor-Tillich can be implemented as two vectorial versions starting from $\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right)$:

$$H_{ZT}\left(P_n(X), m_i\right) = \begin{pmatrix} H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right), m_i\right) \\ H_{ZT}^{vec}\left(P_n(X)||\left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right), m_i\right) \end{pmatrix}.$$

This structure can be exploited to distribute the computation of the first round of ZEST on a long message among N computing units. The messages are divided into $N+1$ blocks of equal sizes $m_0||m_1||...||m_N$, the computation of $H^{vec}_{ZT}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m_0||m_1\right)$ is given to the first unit and the computations of both $H^{vec}_{ZT}\left(P_n(X)||\left(\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right), m_i\right)$ and $H^{vec}_{ZT}\left(P_n(X)||\left(\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right), m_i\right)$ are given to the ith unit (see Figure 9.4).

The exploitation of the parallelism has two costs: first, the total computation cost of the N computing units is $\frac{2N-1}{N}$ times the computation cost of one single unit in a serial mode (it is nearly doubled when N is large). Second, $N-1$ vector by matrix multiplications must be performed at the end to combine the partial hash values. Each of these vector by matrix products requires 4 full modular multiplications and 2 additions. As the cost of a modular multiplication in $\mathbb{F}^*_{2^n}$ is about $2n$ additions [87] (a bit less if an advanced algorithm like Schönhage or Karatsuba is used), the time required to compute the ZEST hash value of a message of length μ using this method is roughly

$$2Lt_{XOR}\left[\frac{2\mu}{N+1} + (4n+1)(N-1) + 2n\right] \tag{9.2}$$

where like in Section 9.4.1, L is the number of words needed to store n bits and t_{XOR} is the time needed to compute the XOR of two words.

The time required to compute a ZEST hash value in parallel is minimized when $N \approx \sqrt{\frac{\mu}{2n}}$. For large messages (like data disks of 100GB$\approx 2^{40}$), this would require too many computing units in practice hence the time is essentially inversely proportional to $N+1$.

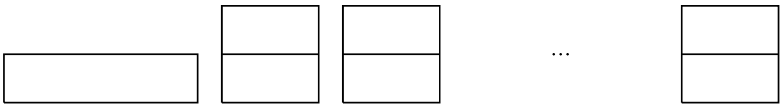

Figure 9.4: Distributing computation among N computing units

A second kind of parallelism can be exploited in ZEST software computation. ZEST computes XORs, SXORs and SHIFTs on bitstrings of length n which in ANSI C are decomposed into corresponding instructions on words of length 32 or 64 bits. Using the SIMD instructions (Single Instruction, Multiple Data) that are commonly found on modern microprocessors, the function computation would be considerably sped up as 128 bits would be treated in parallel. If the implementation is performed using recent graphical accelerators with 512-bit data BUS, then even the computation of ZesT-509 will require only 2 XOR or SXOR per bit on average.

9.5 Adding ZesT into NIST's cooking pot

ZesT has very interesting properties from both security and efficiency points of view. It is provably collision resistant, arguably non-malleable, reasonably efficient in software, very efficient in hardware with respect to both high speed and low area metrics and it is parallelizable. However, it does not completely fulfil NIST's requirements in its call for a new hash function standard [1]. In particular, ZesT is a keyed hash function, it has suboptimal preimage, second preimage and collision resistances and it cannot be used with the output sizes required by NIST because of subgroup attacks.

In this section, we present a fixed-key variant of ZesT and we study its pseudorandom behavior. We then propose modifications of the function in order to satisfy all NIST's requirements and we discuss the efficiency and security implications of these changes. The resulting function is less efficient and considerably less simple than ZesT, but it satisfies all NIST's requirements. The various changes suggested in this section can be added independently into ZesT's recipe; we suggest the reader to select his first-rate of ingredients to satisfy his personal taste for security, efficiency, simplicity and conformity with the NIST's requirements.

9.5.1 Fixing all parameters

Although the formal notion of collision resistance requires defining a family of hash functions with a key as parameter, all standardized hash functions are unkeyed functions. In this section, we suggest a default procedure for fixing the key value of ZesT for any prime value of n up to reasonably large values.

The key of ZesT is made of n, of an irreducible binary polynomial $P_n(X)$ of degree n, and of an initial vector $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)$. Besides the primality constraint on n, our choice of parameter must be clearly non-cheating because of the trapdoor attacks possible for the person who chooses $P_n(X)$ or $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)$. A traditional solution to this problem would be to use a cryptographic hash function (for example SHA-1) and a universal constant (for example pi) to generate the parameters. We suggest a different approach based on an LFSR, that might present an advantage in area constrained applications.

Our goal is to select a key value that is "reasonably random" (say apparently random for Joe-the-Plumber) and obviously non-cheating, and that can be recomputed efficiently to avoid its storage in memory constrained environments. To generate the $3n$ bits of $P_n(X)$ and $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$, we use the maximal Fibonacci LFSR defined by the polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$. As n must be at least 1024 to provide the 512 bits of preimage resistance required by NIST for sensitive applications [1], the degree of the LFSR polynomial must be at least 12. We chose a polynomial of degree 16 because it might help in some applications if the LFSR register has an exact number of bytes. The polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$ has a maximal period 65535; it is the polynomial of degree 16 proposed in [270].

From this LFSR we construct the following (cryptographically weak but good enough for our purpose) pseudorandom bit generator that outputs the first bit of the state.

```
uint16_t reg = INIT ;
for (i =1;i<=65535;i++)
{
    bit = (reg & 0x0001)^((reg & 0x0002) >> 1)
          ^((reg & 0x4000) >> 14) ;
    reg = (reg >> 1) | (bit << 15);
    cout<<bit;
}
```

Let us write $g(INIT, i)$ for the ith bit output by this generator if the state is initially set to INIT. For any n value, we generate our parameters $P_n(X)$ and c as follows:

1. Fix INIT to the binary representation of pi that is 1100100100001111.
2. Build a polynomial from g as follows

   a Set $P_n(X) = X^n + 1$

   b Set the i^{th} bit of $P_n(X)$ to the value $g(INIT, i)$
3. Check wether $P_n(X)$ is irreducible. If yes go to point 6.
4. If $P_n(X)$ is not irreducible, modify it as follows:

   $$P_n(x) \leftarrow X^{-2}\left(P_n(X) + (P_n(X) \bmod X^2)\right) + X^n + 1 + bX^{n-1}$$

   where b is the next bit output by the LFSR.
5. Check wether $P_n(X)$ is irreducible. If yes go to point 6, otherwise go back to point 4.

Table 9.9: Tests implemented into Dieharder version 2.28.1 [5]. The tests preceded by "rft" are ready for testing (the test may - or may not - work correctly) in that version, the tests preceded by "sus" are suspect (they consistently fail "good" generators) and the tests preceded by "dev" are under development.

```
           Diehard Tests                                            RGB Tests
  -d 1  Diehard Birthdays test                              -r 1 RGB Timing test (times the rng)
  [sus: -d 2  Diehard Overlapping Permutations test]        -r 2 RGB Bit Persistence test
  -d 3  Diehard 32x32 Binary Rank test                      -r 3 RGB Ntuple Bit Distribution test suite (-n ntuple)
  -d 4  Diehard 6x8 Binary Rank test                        -r 4 RGB Generalized Minimum Distance test
  -d 5  Diehard Bitstream test                              -r 5 RGB Permutations test (new, partial
  -d 6  Diehard OPSO test                                                            replacement for operm tests)]
  -d 7  Diehard OQSO test                                   [rft: -r 6 RGB Lagged Sums test
  -d 8  Diehard DNA test                                          (do not use the following as tests yet)
  -d 9  Diehard Count the 1s (stream) test                  [dev: -r 7 RGB L-M-Ntuple Distribution test suite
  -d 10 Diehard Count the 1s (byte) test                                                                 (quite long)]
  -d 11 Diehard Parking Lot test                            [dev: -r 8 RGB Overlapping Permutations test]
  -d 12 Diehard Minimum Distance (2D Spheres) test
  -d 13 Diehard 3D Spheres (minimum distance) test             Statistical Test Suite (STS)
  -d 14 Diehard Squeeze test                                -s 1 STS Monobit test
  [sus: -d 15 Diehard Sums test]                            -s 2 STS Runs test
  -d 16 Diehard Runs test                                   -s 3 STS Serial test
  -d 17 Diehard Craps test
  -d 18 Marsaglia and Tsang GCD test                                 User Tests
  [dev: -d 19 Marsaglia and Tsang Gorilla test]             -u 1 User Template (Lagged Sum Test)
```

6. Take the $2n$ following consecutive bits of the LFSR to define the initial vector $(a_0 \; b_0)$, from the degree 0 coefficient of b_0 to the degree $n-1$ coefficient of a_0.

A C++ implementation of this procedure is given in Appendix D.

9.5.2 Use of unkeyed ZesT in standardized applications

In Section 9.3.3 we argued that any malleability property that was present in the Zémor-Tillich hash function and its vectorial variant was removed in ZesT. In this section, we use the Dieharder suite of pseudorandom tests [5] to provide further evidence that ZesT has no apparent weakness. The tests included in the version 2.28.1 of Dieharder we used are listed in Table 9.9.

Each test returns a p-value and a diagnostic according to this p-value. The test is considered as PASSED if the p-value returned is larger than 5%. The byte sequence analyzed is considered as POTENTIALLY WEAK if the p-value is between 1% and 5%, and as POOR if it is smaller than 1%. Many of these tests are probabilistic. On perfectly random bits they will return POTENTIALLY WEAK 4% of the times and POOR 1% of the times.

ZesT was used in a counter mode with n in 31, 61, 127, 157, 223, 251, 383, 509, 1021 and the parameters fixed in the previous section. More precisely, we computed the hash values of $1, 2, ..., 1.000.000$, we truncated them by a few bits to fit perfectly into bytes, we concatenated the results and we analyzed the resulting byte sequence with the Dieharder suite.

The results are shown in Table 9.10. Each column is a version of ZEST labeled by its parameter n and each row is a test of Dieharder labeled as in Table 9.9. As some tests are repeated, each entry shows the number of tests for which the result was PASSED, POTENTIALLY WEAK and POOR. The results for ZEST are very similar to what we may expect from random bytes: for each n value, a few tests are failed below 5% and very few tests are failed below 1%.

Table 9.10: Results of Dieharder for ZEST. The tests are identified as in Table 9.9. As some tests are repeated, each entry gives the number of tests whose verdicts were PASSED, POTENTIALLY WEAK and POOR. a=PASSED (p-value larger than 5%); b=POTENTIALLY WEAK (p-value between 1% and 5%); c=POOR (p-value smaller than 1%).

Test	31	61	127	157	223	251	383	509	1021
- r 3	31a1b	11a1c	11a1b	11a1b	11a1b	12a	12a	12a	11a1b
- r 4	4a	4a	3a1b	4a	4a	4a	4a	3a1b	3a1b
- r 5	5a1c	5a1c	6a	6a	5a1b	6a	6a	6a	6a
- r 6	33a	33a	33a	31a2b	33a	33a	32a1b	31a1b1c	33a
- d 1	1a	1a	1a	1a	1a	1a	1a	1b	1a
- d 3	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 4	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 5	1a	1a	1b	1a	1a	1a	1a	1a	1a
- d 6	1c	1a	1a	1a	1a	1b	1a	1c	1a
- d 7	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 8	1a	1a	1a	1b	1a	1a	1a	1a	1a
- d 9	1a	1b	1a	1a	1a	1a	1a	1a	1a
- d 10	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 11	1a	1a	1a	1a	1a	1a	1a	1b	1a
- d 12	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 13	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 14	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 16	2a	2a	2a	2a	2a	2a	2a	2a	2a
- d 17	2a	2a	2a	2a	2a	2a	2a	2a	2a
- d 18	2a	2a	2a	2a	2a	2a	2a	2a	2a
- s 1	1a	1a	1a	1a	1a	1a	1a	1a	1a
- s 2	1a	1a	1a	1a	1a	1a	1a	1a	1a
- s 3	29a1b	29a1b	30a	29a1c	30a	28a1b1c	30a	30a	27a3b
- u 1	1a	1a	1a	1a	1a	1a	1a	1a	1a

We point out that Dieharder results are insufficient to definitely assert the "pseudorandom behavior" of any function, but they may provide a first positive evidence for it. In the case of ZEST, the results we obtain and the absence of any apparent structure suggest that the unkeyed version may be used in all standardized applications including DSA [12], key derivation [7], HMAC [15] and random bit generation [8].

9.5.3 Reaching optimal (provable) collision resistance

ZEST-n is provably collision resistant up to $n/2$ bits but its output has $2n$ bits. Optimal collision resistance can be reached by replacing the vectorial version of Zémor-Tillich by its projective version in the second round. However, this change has non-negligible efficiency and portability costs.

Recall from Chapter 5 that the projective and vectorial versions of Zémor-Tillich have the same collision resistance if n is not too large. We might therefore define

$$\begin{aligned}&\textsc{ZesT}_1(P_n(X)||\left(\begin{smallmatrix}a_0 & b_0\end{smallmatrix}\right), m)\\ &\qquad := H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix}a_0 & b_0\end{smallmatrix}\right), m||H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix}a_0 & b_0\end{smallmatrix}\right), m\right)\right).\end{aligned}$$

ZEST$_1$ returns projective points $[a:b] \in \mathbb{P}^1(\mathbb{F}_{2^n})$ that can be represented in slightly more than n bits. Following the same reasoning as for ZEST, the security properties of ZEST$_1$ are easily derived. ZEST$_1$ has provable preimage, second preimage and collision resistance up to $n/2$ bits based on the hardness of finding collisions for H_{ZT}^{proj} (for parameters not too large, based on the hardness of finding collisions for H_{ZT}). This is optimal for collision resistance but suboptimal for preimage and second preimage resistance. The actual second preimage security of ZEST$_1$ is indeed not larger than $n/2$ bits but its preimage resistance is arguably as large as n bits because meet-in-the-middle attacks are impossible. Finally, ZEST$_1$ has no particular malleability weakness because the existence of such a weakness would imply the existence of a corresponding weakness in ZEST.

ZEST$_1$ has 2^n+1 possible outputs while it would be more convenient if the output could fit into n bits. For this reason, we suggest the following additional change in ZEST:

$$\textsc{ZesT}_2(P_n(X)||\left(\begin{smallmatrix}a_0 & b_0\end{smallmatrix}\right), m) := \pi\left(a_0, \textsc{ZesT}_1(P_n(X)||\left(\begin{smallmatrix}a_0 & b_0\end{smallmatrix}\right), m)\right)$$

where $\pi : \mathbb{F}_{2^n} \times \mathbb{P}^1(\mathbb{F}_{2^n}) \to \{0,1\}^n$ is defined by

$$\pi(a_0, [a:b]) = \begin{cases} b/a & \text{if } a \neq 0, \\ b_0/a_0 & \text{if } a = 0 \text{ and } a_0 \neq 0, \\ X & \text{if } a = 0 \text{ and } a_0 = 0, \end{cases}$$

for $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) := \textsc{Zest}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)$.

$\textsc{Zest}_2$ is collision resistant if H_{ZT}^{proj} is collision resistant and for parameters not too large, if and only if H_{ZT} is collision resistant. Indeed, let (m, m') be a collision for $\textsc{Zest}_2$: then either the $\textsc{Zest}_1$ hash values of m and m' are the same, either the $\textsc{Zest}_1$ hash value of one of them (let us say m) is equal to $[a_0 : b_0]$ (if $a_0 \neq 0$) or to $[1 : X]$ (if $a_0 = 0$). Defining $h := H_{ZT}^{vec}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)$, the message $m||h$ (if $a_0 \neq 0$) or $m||h||0$ (if $a_0 = 0$) collides with the void message for H_{ZT}^{proj}.

The function $\textsc{Zest}_2$ is less efficient than $\textsc{Zest}$ due to the final division b/a that must be performed most of the times. Divisions in $\mathbb{F}_{2^n}^*$ can be done either with extended versions of the Euclidean algorithm or with a modular exponentiation. Algorithm 7.1 in [87] performs a division in the field with about $4n$ additions. Therefore, the time cost of performing this division is roughly the time needed to process $2n$ bits of the vectorial Zémor-Tillich. For long messages, the division time represents a small overhead but for short messages it will be significant.

The division has another significant drawback: implementations become much more complex. Although the division can be decomposed into additions, the XOR gates of our implementations cannot be reused for the division: this would require additional control logics, which would be as expensive as simply duplicating the XOR gates. The control part of both high-speed and low-area implementations will also considerably increase. Finally, the maximal frequency of the circuit is likely to decrease. The hardware performances of $\textsc{Zest}_2$ should therefore be further examined in the future. In some applications, it might be interesting to compute $\textsc{Zest}$ in hardware and derive the $\textsc{Zest}_2$ value in software.

9.5.4 Reaching optimal (heuristic) second preimage resistance

The second preimage resistance of $\textsc{Zest}$, $\textsc{Zest}_1$ and $\textsc{Zest}_2$ is limited to the collision resistance level. Second preimages can be computed at the price of collisions for the first round, and they suffice to compute second preimages on the whole function. To reach an optimal level of second preimage resistance, distinct n values may be used in the first and second round, with a value about twice as large in the second round as in the first round.

For primes n and $n' \approx 2n$, irreducible polynomials $P_n(X)$, $P_{n'}(X)$ and initial vectors $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \in \mathbb{F}_{2^n}^2 \setminus \{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)\}$, $\left(\begin{smallmatrix} a_0' & b_0' \end{smallmatrix}\right) \in \mathbb{F}_{2^{n'}}^2 \setminus \{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)\}$,

$$\begin{aligned} &\textsc{Zest}_3(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)||P_{n'}(X)||\left(\begin{smallmatrix} a_0' & b_0' \end{smallmatrix}\right), m) \\ &\qquad := \pi\left(a_0, H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m||H_{ZT}^{vec}\left(P_{n'}(X)||\left(\begin{smallmatrix} a_0' & b_0' \end{smallmatrix}\right), m\right)\right)\right). \end{aligned}$$

The collision and preimage resistances of ZEST$_3$ are identical to the previous functions. Moreover, we argue that ZEST$_3$ has optimal second preimage resistance. Indeed, let us suppose that on input m, there exists an algorithm that finds m' colliding with m for ZEST$_3$. Either m and m' have the same intermediate hash value after the first round, either not. The first case clearly reduce to the second preimage resistance of H_{ZT}^{vec} in the first round, which has a complexity of $n'/2 \approx n$ bits. The second case clearly gives a collision in the second round, but this argument only provides a security level of $n/2$ bits.

The second preimage resistance cannot be proved up to n bits based on the balance problem, but heuristic arguments fill in the gap. Indeed, a proof must assume that given m, it is infeasible to find m' such that the vectorial Zémor-Tillich hash values of m and m' are collisions for the projective Zémor-Tillich hash function. This assumption is actually a non-malleability assumption on the vectorial Zémor-Tillich function. We know that the last function is not non-malleable with respect to relations involving concatenations, but the relation here seems completely unrelated to the hash structure and the assumption therefore seems reasonable.

ZEST$_3$ is three times less efficient than ZEST because a vectorial hash value of m is computed with a polynomial of degree $n' \approx 2n$ in the first round and with a polynomial of degree n in the second round. The software time performances, the high speed performances and the lightweight performances will therefore decrease roughly by a factor 3.

9.5.5 Tweaking the function for NIST's output sizes

To avoid subgroup attacks (Section 5.3.5), the parameter n must be prime in ZEST and its variants. However, standardized hash functions usually have output sizes multiple of 32 or even 64. In particular, the output sizes required by NIST for the SHA-3 standard are 224, 256, 384 and 512 [1]. In this section, we propose two alternative modifications of the function to reach these output sizes and we discuss their respective advantages and drawbacks. For each of these alternatives, we define four instances of the function that we call kumquat, lemon, orange and grapefruit, corresponding to the NIST output sizes.

The first alternative chooses n as the smallest prime larger than the targeted size and truncates the result by a few bits. The second alternative chooses n as the largest prime smaller than the targeted size, adds a third round to the function and concatenates the result of the third round with a few bits coming from the second round. The resulting parameters n and the numbers of truncated or added bits ϵ for NIST's output sizes are shown in Table 9.11.

Table 9.11: Parameters for kumquat, lemon, orange and grapefruit in the two alternatives

Name	Output	Truncate		Extend	
		n	ϵ	n	ϵ
kumquat	224	227	3	223	1
lemon	256	257	1	251	5
orange	384	389	5	383	1
grapefruit	512	523	11	509	3

Both changes have (small) negative impact on the efficiency of the function. The first variant uses a larger state and will require an additional word XOR for each addition. For kumquat and in 32-bit architecture, this is not negligible as it represents 14% additional XORs. In hardware, the effect may be considered as negligible. The second variant has a third round hence it requires about $2n$ additional full additions. This effect is negligible for long messages but not for short ones.

The first alternative is more natural and conceptually simpler. However, the collision resistance of a hash function does not imply that the function remains collision resistant if some of its bits are truncated. Of course, we believe that the resulting function is still collision resistant: the opposite would contradict in a strong sense our intuition that ZEST has no non-random behavior. Nevertheless, with this alternative we lose the benefit of provable security for collision resistance. On the other hand, preimage and second preimage resistance (up to the birthday bound) still follow from the hardness of the balance problem.

The second variant is a little more elaborate but its preimage, second preimage and collision resistances are implied by the collision resistance of ZEST. A third round is added in this function because the output of the first round is not random enough to be used for the ϵ additional bits. The function is provably preimage, second preimage and collision resistant up to $n/2$ bits which for collision resistance is only $\epsilon/2$ bits worse than the birthday bound.

9.6 Open problems in the design's parameters

In this section, we discuss possible improvements in ZesT that require further study.

9.6.1 Use of special polynomials

Special polynomials, in particular sparse polynomials, may significantly improve the efficiency of ZesT in software [207]. Indeed, if $P_n(X) = X^n + P_{31}(X)$, a modular reduction only requires one word XOR instead of L word XORs. For large n and small architectures, this results in an efficiency improvement of nearly 25%. The efficiency in hardware, at least for our designs of Section 9.4.2 and 9.4.3, is unchanged. However, the use of sparse polynomials impacts the pseudorandomness behavior of ZesT as the bit mixing is achieved through the modular reductions.

We heuristically observed that "the images of messages with a lot of zeroes also have a lot of zeroes". This fact may be explained from the properties of the polynomials $f_i(X)$ and of matrix powers that we discussed in Section 5.1.3: the polynomial $f_{2^i}(X) = X^{2^i} + 1$ has a lot of zeroes in its representation for any i. Similarly, we observed that "the images of messages with a lot of ones have a lot of zeroes".

Further study should demonstrate whether this weakness can be turned into an actual collision attack against the Zémor-Tillich hash function when sparse polynomials are used, or if its damage is limited to pseudorandom properties. In the second case, we solved the problem in [208] with an appropriate intermediate permutation between the first and the second rounds of ZesT. The purpose of this permutation is to mix the bits produced after the first round; it may be as simple as an XOR by a constant whose bits do not follow any repetition pattern, for example the bits of pi [208].

9.6.2 Other graph generators

From both efficiency and security points of view, the generators chosen in the Zémor-Tillich hash function are better than other Cayley hash proposals like LPS and Morgenstern [68, 259, 204]. There may exist other generators sets that still improve the function, but further study is required to assert their security. Balance and representation problems are not classical problems but in the case of the Zémor-Tillich hash they have at least been studied from 15 years. If the generators change, the confidence that we may have on the Zémor-Tillich hash function will not transfer automatically to the new function because the hardness of these problems seems to depend a lot on the generators choice.

From an efficiency point of view, the group $SL(2,\mathbb{F}_{2^n})$ is definitely better than the group $SL(2,\mathbb{F}_p)$ used in [68]. The function efficiency will however improve a lot if sets of 4, 8 or 16 generators can be used instead of the two generators A_0 and A_1 of the Zémor-Tillich hash function.

We may also try to change the generators to protect against the trapdoor attack on the vectorial version. The choice of A_0 and A_1 was particularly unlucky with this respect, because the attack will likely be unpractical for randomly chosen generators. Indeed, let $M, M' \in SL(2,\mathbb{F}_p)$ satisfy $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) M = \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) M'$. This is equivalent to $\det(M+M') = 0$, hence to $\det(I + M'M^{-1}) = 0$ and finally to $\mathrm{Tr}(M'M^{-1}) = 0$. There are about 2^{3n} matrices in $SL(2,\mathbb{F}_p)$, among which about 2^{2n} matrices with 0 trace, hence for randomly chosen generators we would need about 2^n random products of them in order to find two matrices of the correct form. Choosing as generators the two matrices

$$A'_0 = \begin{pmatrix} X^2 & 1 \\ 1 & 0 \end{pmatrix} \qquad A'_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}.$$

seems safe with respect to the vectorial trapdoor attack. The change of X by X^2 in A_0 will not affect too much the efficiency. Moreover, this change would have the advantage to lessen greatly the density of the subset Ω generated in $SL(2,\mathbb{F}_2[X])$, which could be benefic against lifting attacks.

Since balance and representation problems in groups $SL(2,.)$ are badly known in general and since their hardness seems to depend a lot on the choice of generators, any change in the generators should be followed by its own careful security study.

9.6.3 Number of rounds

Heuristic reasoning and statistical tests on the outputs of ZesT used in a counter mode tend to assert that two rounds are enough for its security as a general-purpose hash function. However, statistical tests can only give a first answer for pseudorandomness and we might have missed some way to exploit the group structure or even some hidden quasi-group structure in the function. In particular, we believe that the number of rounds necessary in ZesT should be better examined, especially in the light of standard attacks against traditional hash functions. We leave this question as an interesting open problem.

9.7 Conclusion

In this chapter, we transformed a provable hash function with its inherent malleability weaknesses into a practical, all-purpose hash function. We started from the Zémor-Tillich hash function because of its elegant Cayley hash design, its potentially great efficiency and the possibility to parallelize the hash computation. Although the collision resistance of this function relies on a problem that is not classical in cryptography, the problem has resisted 15 years of cryptanalytic attempts and we believe that it deserves broader interest in the cryptographic community.

We called ZesT our modification of the Zémor-Tillich hash function. ZesT is provably collision, preimage and second preimage resistant. Our first implementations show that it is reasonably fast in software and efficient in FPGA and that it admits ultra-lightweight implementations. In particular, it is only 4 to 10 times as slow as SHA in software, comparable to SHA on FPGA and better than any other known hash function for area constrained applications. Moreover, the hash computation can be distributed without affecting the result.

The ZesT function does not fill all NIST requirements in its recent call, but it can be easily modified to comply with those requirements. As long as the security of the Zémor-Tillich hash function is trusted, ZesT is definitely an interesting hash candidate as an all-purpose hash function.

Chapter 10

Conclusion and open problems

Hash functions are essential cryptographic primitives, useful for message authentication codes and digital signatures but also for many and very diverse other cryptographic protocols. As the main properties required from hash functions are collision and (second) primage resistances, it is very desirable to build these properties on the hardness of mathematical problems that can be studied independently by mathematicians and cryptographers.

The expander hash design, proposed in the early nineties by Gilles Zémor and Jean-Pierre Tillich and recently rediscovered by Denis Charles, Eyal Goren and Kristin Lauter, provides "provable security" for preimage and collision resistance in all existing cases. Besides, it has significant other desirable properties but also some weaknesses following from its important mathematical structure.

The goal of this thesis was to formalize and prove general properties of expander hashes, to investigate the actual security and efficiency of existing constructions and to provide solutions for the inherent weaknesses of the design.

This chapter concludes our tour of expander hashes. In Section 10.1 we summarize the content of the thesis, in Section 10.2 we point out its main contributions and in Section 10.3 we discuss important problems opened or left open by our work.

10.1 Expander hash functions

As traditional hash function designs are being questioned, the simple and clear design of expander hashes deserves a renewed interest today. In expander hashes, the main properties of hash functions, their collision, second preimage and preimage resistances and their output distributions, can be interpreted and studied with the graph-theoretic notions of girths, cycles and expanding constants. In the important case of Cayley hashes, collision, second preimage and preimage resistances admit further interpretation and study in terms of group-theoretical problems. Although these problems are not classical in cryptography, in some cases they have resisted cryptanalysis attempts for more than 15 years and this alone justifies further consideration.

Besides, expander hashes have significant advantages over traditional hash functions. No padding nor domain-extension transform are required as expander hashes already process message of arbitrary sizes. Cayley hash functions can be computed in parallel, a property whose interest for efficiency cannot be overstated while the future of computer architectures is clearly in the parallelism. If a graph family is used, the function is scalable, which means that it may reach any level of security with appropriate parameter choices. The main drawback of expander hashes is their malleability properties induced by their mathematical structure.

The Zémor-Tillich hash function was one of the first expander hash proposals. Its computation is the most efficient among all expander hashes as it only requires a few additions per bit in a field of characteristic 2. The function is parameterized by a small integer n and an irreducible binary polynomial $P_n(X)$. Existing attacks have shown that the integer n should be prime and large enough to prevent birthday attacks in a set of size 2^n. The polynomial must be fixed in an unambiguous way to avoid trapdoor attacks. Although the output of Zémor-Tillich is a 2×2 matrix whose coefficients have n bits, its output set only has about 2^{3n} elements. Moreover, the exact security for collision, second preimage and preimage resistances is of $n/2$ bits instead of the optimal $3n/2$, $3n$ and $3n$ bits. The function suffers from malleability and it is invertible on short messages, but it has remained essentially unbroken since 1994. Two variants, the vectorial and the projective variants, have reduced outputs and the same security against collision and preimage finding algorithms.

The use of LPS and Morgenstern Ramanujan graphs in the expander hash design was motivated by their optimal expansion with respect to other undirected graphs. This advantage must however be moderated by the fact that directed graphs, like the Zémor-Tillich graphs, may still have a better expansion rate than optimal undirected graphs. Moreover, LPS and Morgenstern hash functions are now completely broken: efficient probabilistic polynomial time algorithms exist both for finding collisions and for computing preimages. Both functions may be repaired easily but at the cost of some efficiency and the Ramanujan property. The attacks were actually made possible by the mathematical structure necessary to reach the Ramanujan expansion rate in

LPS and Morgenstern graphs, hence these two examples suggest that using Ramanujan graphs in the expander hash design may not be a good idea.

This conclusion is qualified with Pizer hashes that use the Ramanujan graph family of Pizer and have kept resisting all attacks so far. Pizer graphs admit two equivalent descriptions, one in the language of orders in a quaternion algebra and another one in the language of supersingular elliptic curves. In the first language, Pizer hashes would be broken by techniques similar to those breaking LPS and Morgenstern hash functions. However, Pizer hashes were defined in the language of supersingular elliptic curves and the translation from one language to the other is not efficient. Pizer hashes might actually be the most secure expander hash function proposed so far. The function has however little chance to be used once in practice due to its poor efficiency and to remaining issues in the key generation algorithm.

Expander hash functions are malleable: it is possible to relate two hash values without knowing the corresponding messages. The property does not contradict collision nor preimage resistances but it prevents from using the hash function in wider contexts. This malleability can be captured through Canetti et al.'s correlation intractability definition or through Boldyreva et al.'s non-malleability definition. Depending on the protocols, malleability properties in general and in the particular case of expander hashes may or may not create security threats. In some cases like Cayley hashes, the malleability properties can be turned into a major advantage of the function.

The malleability properties of expander hashes may be easily removed with a little additional design. The ZesT hash function is essentially the vectorial version of Zémor-Tillich iterated twice as in HMAC. The function is provably collision and preimage resistant if the balance problem corresponding to Zémor-Tillich is hard. It is between 4 and 10 times slower than SHA in software but it has comparable efficiency on FPGA and it admits an ultra-low weight implementation in ASIC. The function computation can be parallelized greatly and efficiently, and its simplicity will certainly allow for a much wider range of implementations and for software-optimized code generation. A careful examination and pseudorandom tests performed with the Dieharder revealed no apparent malleability weakness, which suggests that the function can be used as a general-purpose hash function. Moreover, ZesT can be slightly modified to reach all requirements of the NIST competition. ZesT will really become practical if the hardness of the balance problem corresponding to Zémor-Tillich becomes better established.

10.2 Contributions of the thesis

The goal of this thesis was to formalize and prove general properties of expander hashes, to investigate the actual security and efficiency of existing constructions and to provide solutions for the inherent weaknesses of the design.

Our first main contribution in the thesis is a security review of the Zémor-Tillich hash function, new collision and preimage attacks against it and the introduction of its vectorial and projective variants. Some authors had been claiming attacks against the Zémor-Tillich hash function but the exact implications of these attacks was not clear to the community and the function was sometimes wrongly considered as broken. In Chapter 5, we described, analyzed and in many cases improved attacks that had often only been justified by concrete examples on particular parameters or on reduced versions. Subsequently, we introduced new attacks on the function and two variants, the vectorial and projective variants, that have reduced output sizes for essentially the same security[1].

Our second main contribution is the full cryptanalysis of the LPS and Morgenstern hash functions[2] presented in Chapter 6. Tillich and Zémor computed collisions for the LPS hash functions; we found a non-trivial extension of their algorithm to a preimage algorithm and could extend both algorithms to the Morgenstern hash function as well.

Our third main contribution is the introduction of ZesT, an all-purpose hash function based on the Zémor-Tillich hash function[3] presented in Chapter 9. ZesT keeps the best properties of Zémor-Tillich and avoids its main weaknesses. Surprisingly for a provable hash function, the ZesT hash function will really become practical once the hardness of the representation problem corresponding to the Zémor-Tillich hash function is better established. ZesT is provably secure, parallelizable, scalable, and admits a wide range of (very) efficient implementations.

This thesis is the first review of the expander hash design. Besides the main contributions mentioned above, a review of hash functions literature is proposed in Chapter 2 and a review of the main "provable" constructions in Chapter 3. In Chapter 4, expander and Cayley hash properties are carefully related with graph and group properties. Some aspects of Pizer hashes are put forward in Chapter 7. Malleability properties of expander hashes

[1] Together with Jean-Jacques Quisquater, Jean-Pierre Tillich and Gilles Zémor [207].

[2] Together with Kristin Lauter and Jean-Jacques Quisquater [204].

[3] Together with Giacomo de Meulenaer, Jean-Jacques Quisquater, Jean-Pierre Tillich, Nicolas Veyrat-Charvillon and Gilles Zémor [208, 88, 203].

are investigated in Chapter 8 together with their possible positive and negative consequences in applications. This thesis covers all aspects of expander hashes, from applications of hash functions to security properties and from the security of particular instances to their practicability and their efficiency in software and in hardware.

Scientific contributions of the author that are not related to this thesis are briefly discussed in Appendix A.

10.3 Open problems

In this section, we describe important problems that have been opened or left open by our work. The interest of studying representation problems and their potential applications to cryptography is justified in Section 10.3.1, the need for new hash functions definitions and designs is stressed in Section 10.3.2, and further problems encountered in the thesis are collected in Section 10.3.3.

10.3.1 Representation problems in cryptography

The Cayley hash function design is a very appealing one. The collision resistance of a Cayley hash is equivalent to the hardness of the corresponding balance problem and it implies the hardness of the corresponding representation problem. In general, this last problem is not strictly equivalent to the balance problem but its simple form makes it simpler to study. Moreover, as the two problems are very similar, it seems unlikely that a new technique could be developed to solve a balance problem without affecting the security of the corresponding representation problem.

In this thesis, we have seen representations problems that are easy to solve and others that have resisted all cryptanalytic attempts. In general, the exact hardness of representation problems is not known and it definitely deserves further study. In particular, it would be very interesting to identify classes of groups and generators such that the corresponding representation problems are easy, other classes that reduce to well-known problems like discrete logarithm problems, and finally classes that cannot be reduced to known problems but still may be thought of as hard problems.

New techniques and ideas must be developed in order to break these problems or to provide good arguments in favor of their hardness. Related problems from the theory of expander graphs and from representation theory are likely to bring new insights if appropriate connections can be found. Completely elementary new techniques might also be able to break many problems; in particular, the approaches we identified in Section 5.6 as promising for the Zémor-Tillich hash function may serve other problems.

The interest of representation problems is not limited to hash functions. They are useful to generate pseudorandom permutations and we suspect that they have an important role to play to prevent key recovery attacks in the encryption scheme of [233] and in many other cryptographic applications that remain to be discovered. A large part of cryptography is built upon the hardness of a few mathematical problems. Although the security of the discrete logarithm, the integer factorization and the elliptic curve discrete logarithm problems are well established, a significant breakthrough might still happen in the resolution of one or all of them. As some instances of representation problems seem to be hard problems, it is a natural and very interesting open problem to see how much of modern cryptography we can build upon them.

10.3.2 Better understanding of hash functions

Even more important for cryptography but less specific to expander hashes, this thesis confirmed the need for new definitions and new designs for hash functions.

New definitions are needed to characterize the malleability properties of provable hash functions and to replace the random oracle model when collision resistance is not sufficient. In 1993, Anderson [27] already pointed out that protocol designers should explicitly tell which properties they assume from the hash functions they use. The solution does not rely on assuming a random oracle model in applications: besides its soundness issues, the model discards nearly all provable hash functions because of their malleability properties, even if they could actually be used in many protocols.

The correlation intractability and non-malleability definitions discussed in Chapter 8 are very appealing; their usefulness in applications should however be further demonstrated. Another interesting recent result is the notion of programmable hash functions [132] that seem to embrace a wide spectrum of definitions, from very useful (non-achievable) notions to achievable but less useful ones.

From the point of view of standardization, special-purpose hash functions that are limited to some applications but would render other applications insecure, are not desirable. The ZesT function that we developed in Chapter 9 presents a good trade-off between provable and heuristic properties; it uses a provable hash function as a building block to construct an all-purpose hash function by removing the original malleability properties with additional design. A similar approach was adopted in the SWIFFTX submission to NIST, based on the provable hash function SWIFFT described in Section 3.3. We believe that this approach is very meaningful and should be further developed in the future.

10.3.3 Miscellaneous

As complete as we tried to be in this thesis, there remain some gaps that we could not fill in the provable security aspects of expander hashes. In Section 4.2.5, despite of some intuition that the result is true when collisions are hard to find, we could not prove nor disprove an equivalent of the "left-over hash lemma" to use expander hashes for randomness extraction. In Section 4.3.2 and 4.3.3, we could not prove nor disprove the expansion property for one of Zémor's construction and for the Zémor-Tillich hash function, although it is widely believed that Cayley graphs of special linear groups are good expanders. In a sense, this last gap might be a good news for the security of these functions: precise bounds on the eigenvalues might come together with a better understanding of the graph structure and hence bring new ideas for collision searches.

10.4 Conclusion

With this chapter, we conclude our tour of expander hashes, a class of hash functions with a particularly elegant design based on graphs and non-Abelian groups. These functions are still in their childhood but in our opinion, they could really provide very interesting NIST hash candidates... for SHA-4.

Bibliography

[1] http://csrc.nist.gov/groups/ST/hash/documents/SHA-3_FR_Notice_Nov02_2007%20-%20more%20readable%20version.pdf.

[2] http://paginas.terra.com.br/informatica/paulobarreto/hflounge.html.

[3] http://www2.mat.dtu.dk/people/S.Thomsen/bib.html.

[4] American National Institute for Standards and Technology (NIST). http://www.nist.gov/.

[5] Dieharder. http://www.phy.duke.edu/ rgb/General/dieharder.php.

[6] European Organization for Nuclear Research (CERN). http://public.web.cern.ch/public/.

[7] Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf.

[8] Recommendation for random number generation using deterministic random bit generators. http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf.

[9] FIPS 46-3 Data Encryption Standard (DES). Federal Information Processing Standards Publication, 1976.

[10] Information technology - security techniques - hash functions, part 1: General and part 2: Hash functions using an n-bit block cipher algorithm. DIS 10118, ISO/IEC, 1992.

[11] Advances in cryptology - CRYPTO '98, 18th annual international cryptology conference, Santa Barbara, California, USA, august 23-27, 1998,

proceedings. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[12] FIPS 186-2 Digital Signature Standard (DSS). Federal Information Processing Standards Publication, 2000.

[13] FIPS 197 Advanced Encryption Standard (AES). Federal Information Processing Standards Publication, 2001.

[14] FIPS 180-2 Secure Hash Standard. Federal Information Processing Standards Publication, 2002.

[15] FIPS 198 The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication, March 2002.

[16] The SHA-3 zoo. `http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo`, 2008.

[17] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Knudsen [152], pages 418–433.

[18] K. S. Abdukhalikov and C. Kim. On the security of the hashing scheme based on SL2. In *FSE '98: Proceedings of the 5th International Workshop on Fast Software Encryption*, pages 93–102, London, UK, 1998. Springer-Verlag.

[19] M. Ajtai. Generating hard instances of lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(7), 1996.

[20] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. In *STOC*, pages 132–140. ACM, 1987.

[21] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[22] N. Alon, I. Benjamini, E. Lubetzky, and S. Sodin. Non-backtracking random walks mix faster. arXiv:math/0610550, October 2006.

[23] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. *Computational Complexity*, 5:60–75, 1995.

[24] N. Alon and V. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, series B*, 38:73–88, 1985.

[25] American Bankers Association. Financial institution message authentication (wholesale). ANSI X9.9 (revised), April 7 1986.

[26] American Bankers Association. Financial institution retail message authentication. ANSI 9.19, August 13 1986.

[27] R. Anderson. The classification of hash functions. IMA Conference on Cryptography and Coding, 1993.

[28] E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Seven-property-preserving iterated hashing: ROX. In *ASIACRYPT*, pages 130–146, 2007.

[29] S. Arora and B. Barak. Complexity theory: A modern approach. `http://www.cs.princeton.edu/theory/complexity/`, 2008.

[30] L. Babai and Ákos Seress. On the diameter of permutation groups. *Eur. J. Comb.*, 13(4):231–243, 1992.

[31] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey, 1995.

[32] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, 2006.

[33] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Lecture Notes in Computer Science*, 1109:1–15, 1996.

[34] M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions—the HMAC construction. *CryptoBytes*, 2, 1996.

[35] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 216–233, London, UK, 1994. Springer-Verlag.

[36] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography and application to virus protection. pages 45–56, 1995.

[37] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[38] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997.

[39] M. Bellare and A. Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Franklin [107], pages 273–289.

[40] M. Bellare and T. Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In *ASIACRYPT*, pages 299–314, 2006.

[41] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. pages 62–73. ACM Press, 1993.

[42] E. Biham and R. Chen. Near-collisions of SHA-0. In Franklin [107], pages 290–305.

[43] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 2–21, London, UK, 1991. Springer-Verlag.

[44] J. Black, M. Cochran, and T. Shrimpton. On the impossibility of highly efficient block cipher-based hash functions.

[45] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.

[46] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block cipher-based hash function constructions from PGV. In Yung [272], pages 320–335.

[47] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography.* Cambridge University Press, New York, NY, USA, 1999.

[48] M. Blum. Coin flipping by telephone. In *CRYPTO*, pages 11–15, 1981.

[49] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin. Hash functions and RFID tags: Mind the gap. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 283–299. Springer, 2008.

[50] A. Boldyreva, D. Cash, M. Fischlin, and B. Warinschi. Non-malleable hash functions. Manuscript, 2007.

[51] A. Boldyreva, D. Cash, M. Fischlin, and B. Warinschi. Foundations of non-malleable hash and one-way functions. Cryptology ePrint Archive, Report 2009/065, 2009. `http://eprint.iacr.org/`.

[52] D. Boneh and M. K. Franklin. Efficient generation of shared RSA keys (extended abstract). In Kaliski Jr. [148], pages 425–439.

[53] J. Bosset. Contre les risques d'altération, un système de certification des informations. *Informatique*, 107, 1977.

[54] S. A. Brands. An efficient off-line electronic cash system based on the representation problem. In *246*, page 77. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1993.

[55] G. Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[56] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[57] E. F. Brickell. Solving low density knapsacks. In *CRYPTO*, pages 25–37, 1983.

[58] J.-Y. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In *FOCS*, pages 468–477, 1997.

[59] P. Camion. Can a fast signature scheme without secret key be secure? In *AAECC*, volume 228 of *Springer Verlag Lecture Notes in Computer Science*, pages 187–196, 1987.

[60] P. Camion and J. Patarin. The knapsack hash function proposed at CRYPTO'89 can be broken. In *EUROCRYPT*, pages 39–53, 1991.

[61] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Kaliski Jr. [148], pages 455–469.

[62] R. Canetti, editor. *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*. Springer, 2008.

[63] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[64] R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *STOC*, pages 131–140, 1998.

[65] J. L. Carter and M. N. Wegman. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM.

[66] J. M. Cerviño. On the correspondence between supersingular elliptic curves and maximal quaternionic orders. `http://arxiv.org/abs/math/0404538v1`, 2004.

[67] F. Chabaud and A. Joux. Differential collisions in SHA-0. In Krawczyk [11], pages 56–71.

[68] D. Charles, E. Goren, and K. Lauter. Cryptographic hash functions from expander graphs. *J. Cryptology*, 22(1):93–113, 2009.

[69] D. Charles and K. Lauter. Computing modular polynomials, 2005.

[70] C. Charnes and J. Pieprzyk. Attacking the SL2 hashing scheme. In *ASIACRYPT '94: Proceedings of the 4th International Conference on the Theory and Applications of Cryptology*, pages 322–330, London, UK, 1995. Springer-Verlag.

[71] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Feigenbaum [97], pages 470–484.

[72] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. In *CHES*, pages 298–310, 2006.

[73] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Cost-efficient SHA hardware accelerators. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 16(8):999–1008, August 2008.

[74] F. Chung. Laplacian and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9:1–19, 2005.

[75] F. Chung. Random walks on directed and undirected graphs. Lecture Notes of Math261, 2005.

[76] S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.

[77] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[78] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In Shoup [247], pages 430–448.

[79] J.-S. Coron, J. Patarin, and Y. Seurin. The random oracle model and the ideal cipher model are equivalent. In Wagner [264], pages 1–20.

[80] M. J. Coster, A. Joux, B. A. Lamacchia, A. M. Odlyzko, C.-P. Schnorr, and J. Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

[81] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

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

[83] I. Damgård. A design principle for hash functions. In Brassard [55], pages 416–427.

[84] G. Davidoff, P. Sarnak, and A. Valette. *Elementary Number Theory, Group Theory, and Ramanujan Graphs.* Cambridge University Press, 2003.

[85] D. Davies and W. Price. Digital signatures, an update. In *5th International Conference on Computer communication*, 1984.

[86] de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie voor Wetenschappen*, 49:758–764, 1946.

[87] G. M. de Dormale. *Destructive and Constructive Aspects of Efficient Algorithms and Implementation of Cryptographic Hardware.* PhD thesis, Université catholique de Louvain, 2008.

[88] G. de Meulenaer, C. Petit, and J.-J. Quisquater. Hardware implementations of a variant of Zémor-Tillich hash function: Can a provably secure hash function be very efficient? Preprint, 2009.

[89] Y. Desmedt, editor. *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*. Springer, 1994.

[90] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[91] H. Dobbertin. Cryptanalysis of MD4. In Gollmann [122], pages 53–69.

[92] H. Dobbertin. RIPEMD with two-round compress function is not collision-free. *J. Cryptology*, 10(1):51–70, 1997.

[93] J. Dodziuk. Difference equations, isoperimetric inequality, and transience of certain random walks. *Transactions of the American Mathematical Society*, 284:787–794, 1984.

[94] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 542–552, New York, NY, USA, 1991. ACM Press.

[95] W. F. Ehrsam, C. H. W. Meyer, R. L. Powers, P. N. Prentice, J. L. Smith, and W. L. Tuchman. Block cipher system for data security. US Patent 3958081, May 1976.

[96] S. Even and O. Goldreich. The minimum-length generator sequence problem is NP-hard. *J. Algorithms*, 2(3):311–313, 1981.

[97] J. Feigenbaum, editor. *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992.

[98] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. pages 357–370. 2004.

[99] M. Feldhofer and C. Rechberger. A case against currently used hash functions in RFID protocols. pages 372–381. 2006.

[100] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[101] M. Fischlin. Perfectly-crafted Swiss army knives (in theory). ECRYPT Workshop on Hash Functions in Cryptology, Theory and Practice, June 2008.

[102] M. Fischlin. Security of NMAC and HMAC based on non-malleability. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2008.

[103] M. Fischlin and A. Lehmann. Security-amplifying combiners for collision-resistant hash functions. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 2007.

[104] M. Fischlin and A. Lehmann. Multi-property preserving combiners for hash functions. In Canetti [62], pages 375–392.

[105] M. Fischlin, A. Lehmann, and K. Pietrzak. Robust multi-property combiners for hash functions revisited. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 655–666. Springer, 2008.

[106] P. Flajolet and M. Soria. Gaussian limiting distributions for the number of components in combinatorial structures. *J. Comb. Theory Ser. A*, 53(2):165–182, 1990.

[107] M. K. Franklin, editor. *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 2004.

[108] J. Friedman. Some geometric aspects of graphs and their eigenfunctions. *Duke Mathematical Journal*, 69(3):487–525, 1993.

[109] S. Galbraith. Constructing isogenies between elliptic curves over nite elds, 1999.

[110] M. Gebhardt, G. Illies, and W. Schindler. A note on practical value of single hash collisions for special file formats. NIST First Cryptographic Hash Workshop,, October/November 2005.

[111] W. Geiselmann. A note on the hash function of Tillich and Zémor. In Gollmann [122], pages 51–52.

[112] D. Giry and P. Bulens. keylength.com.

[113] P. Godlewski and P. Camion. Manipulations and errors, detection and localization. In *EUROCRYPT*, pages 97–106, 1988.

[114] O. Goldreich. *Fundations of Cryptography, Volume II Basic Applications*. Cambridge University Press, 2004.

[115] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42), 1996.

[116] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[117] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[118] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[119] S. Goldwasser and Y. T. Kalai. On the (in)security of the Fiat-Shamir paradigm. In *FOCS*, pages 102–. IEEE Computer Society, 2003.

[120] S. Goldwasser, S. Micali, and R. L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *FOCS*, pages 441–448. IEEE, 1984.

[121] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.

[122] D. Gollmann, editor. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*. Springer, 1996.

[123] F. Gosset, F.-X. Standaert, and J.-J. Quisquater. FPGA implementation of SQUASH. In *Proceedings of the 29th Symposium on Information Theory in the Benelux*, 2008.

[124] L. C. Guillou and J.-J. Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In S. Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 1988.

[125] S. Hada and T. Tanaka. On the existence of 3-rounds zero-knowledge protocols. In Krawczyk [11], pages 408–423.

[126] I. Haitner and O. Reingold. Statistically-hiding commitment from any one-way function. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 2007. ACM.

[127] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 1996.

[128] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.

[129] H. A. Helfgott. Growth and generation in $SL_2(Z/pZ)$, 2005.

[130] A. Herzberg. On tolerant cryptographic constructions. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 172–190. Springer, 2005.

[131] S. Hirose. Some plausible constructions of double-block-length hash functions. In M. J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2006.

[132] D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. In Wagner [264], pages 21–38.

[133] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.

[134] B. Huppert. *Endliche Gruppen I.* Springer-Verlag, 1967.

[135] H. Imai and Y. Zheng, editors. *Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, volume 1751 of *Lecture Notes in Computer Science*. Springer, 2000.

[136] R. Impagliazzo. A personal view of average-case complexity. *10th Annual Structure in Complexity Theory Conference (SCT'95)*, page 134, 1995.

[137] R. Impagliazzo. CSE 208 class notes. `http://www-cse.ucsd.edu/~russell/RANDCLASS/outline.html`, 2001.

[138] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *FOCS*, pages 236–241. IEEE, 1989.

[139] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. pages 44–61, 1989.

[140] K. Ireland and M. Rosen. *A classical introduction to modern Number Theory*. Springer Verlag, 1993.

[141] ISO. Banking – approved algorithms for message authentication, Part 1, DEA, Part 2, Message Authentication Algorithm (MAA). ISO 8731:1987, 1987.

[142] ISO/IEC. Information technology data cryptographic techniques data integrity mechanisms using a cryptographic check function employing a block cipher algorithm. ISO/IEC 9797:1993, 1993.

[143] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, pages 143–154, 1996.

[144] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theor. Comput. Sci.*, 36(2-3):265–289, 1985.

[145] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Franklin [107], pages 306–316.

[146] A. Joux and J. Stern. Improving the critical density of the Lagarias-Odlyzko attack against subset sum problems. pages 258–264, 1991.

[147] M. Joye. Introduction élémentaire à la théorie des courbes elliptiques. Technical report, UCL CRYPTO group, 1995.

[148] B. S. Kaliski Jr., editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997.

[149] M. Kassabov, A. Lubotzky, and N. Nikolov. Finite simple groups as expanders, 2005.

[150] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[151] L. Knudsen, F. Mendel, C. Rechberger, and S. Thomsen. MDC-2. Ecrypt Workshop on Hash Functions, June 2008.

[152] L. R. Knudsen, editor. *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*. Springer, 2002.

[153] L. R. Knudsen and B. Preneel. Fast and secure hashing based on codes. In Kaliski Jr. [148], pages 485–498.

[154] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[155] H. Krawczyk. LFSR-based hashing and authentication. In Desmedt [89], pages 129–139.

[156] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*. The Internet Society, 2000.

[157] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. 32(1):229–246, Jan. 1985. Preliminary version in *Proc. 24th IEEE Foundations Computer Science Symp.*, pp. 1–10, 1983.

[158] X. Lai and J. L. Massey. Hash function based on block ciphers. In *EUROCRYPT*, pages 55–70, 1992.

[159] L. Lamport. Constructing digital signatures from a one-way function. Technical report, October 1979.

[160] P. Lancaster and M. Tismenetsky. *The theory of matrices*. Computer Science and Applied Mathematics. Academic Press Inc., Orlando, Fla., second edition, 1985.

[161] S. Lang. *Algebra*. Addison-Wesley, 1965.

[162] F. T. Leighton and S. Micali. Secret-key agreement without public-key cryptography. In Stinson [256], pages 456–479.

[163] A. K. Lenstra. Key length, contribution to "The handbook of information security", 2004.

[164] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.

[165] H. W. J. L. L. Lenstra, A. K.; Lenstra. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(5):515–534, 1982.

[166] R. Lidl and H. Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. With a foreword by P. M. Cohn.

[167] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[168] A. Lubotzky and B. Weis. Groups and expanders, 1992.

[169] S. Lucks and M. Daum. The story of Alice and her Boss: Hash functions and the blind passenger attack. Rump Session of Eurocrypt 2005, 2005.

[170] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In I. Wegener, V. Sassone, and B. Preneel, editors, *Proceedings of the 33rd international colloquium on automata, languages and programming - ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155, Venice, Italy, July 2006. Springer-Verlag.

[171] V. Lyubashevsky and D. Micciancio. Asymptotically efficient lattice-based digital signatures. In Canetti [62], pages 37–54.

[172] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably secure FFT hashing. In *NIST 2nd Cryptogaphic Hash Workshop*, 2006.

[173] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In Nyberg [196], pages 54–72.

[174] D. Mackenzie. Computer science: Hash of the future? *Science*, 319(5869):1481+, March 2008.

[175] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM technical Disclosure Bulletin*, 27:5658–5659, 1985.

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

[177] R. Merkle. Secrecy, authentication, and public key systems. Technical report, UMI Research Press, 1979.

[178] R. C. Merkle. One-way hash functions and DES. In Brassard [55], pages 428–446.

[179] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions On Information Theory*, 24:525–530, 1978.

[180] J.-F. Mestre. La méthode des graphes. Exemples et applications. In *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata, 1986)*, pages 217–242, Nagoya, 1986. Nagoya Univ.

[181] D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM J. Comput.*, 30(6):2008–2035, 2000.

[182] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *FOCS*, pages 356–365. IEEE Computer Society, 2002.

[183] D. Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor. *SIAM J. Comput.*, 34(1):118–169, 2005.

[184] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. volume 16, pages 365–411, Basel, Switzerland, Switzerland, 2007. Birkhauser Verlag.

[185] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 372–381, Washington, DC, USA, 2004. IEEE Computer Society.

[186] V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[187] I. Mironov. Hash functions: Theory, attacks, and applications.

[188] S. Miyaguchi, K. Ohta, and M. Iwata. 128-bit hash function (N-hash). Technical report, NTT, Nov. 1990.

[189] R. Montenegro. Conductance and canonical paths for directed non-lazy walks. arXiv:math/0611585v3, January 2008.

[190] M. Morgenstern. Existence and explicit construction of $q + 1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory*, B 62:44–62, 1994.

[191] M. Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[192] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 33–43, New York, NY, USA, 1989. ACM.

[193] W. Nevelsteen and B. Preneel. Software performance of universal hash functions. In *EUROCRYPT*, pages 24–41, 1999.

[194] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Yung [272], pages 111–126.

[195] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

[196] K. Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.

[197] A. Odlyzko. Discrete logarithms: the past and the future. *Designs, Codes, and Cryptography*, 19:129–145, 1999.

[198] K. Ohta and T. Okamoto. On concrete security treatment of signatures derived from identification. In Krawczyk [11], pages 354–369.

[199] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1992.

[200] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[201] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166. Springer, 2006.

[202] C. Peikert and A. Rosen. Lattices that admit logarithmic worst-case to average-case connection factors. In D. S. Johnson and U. Feige, editors, *STOC*, pages 478–487. ACM, 2007.

[203] C. Petit, G. de Meulenaer, J.-J. Quisquater, J.-P. Tillich, N. Veyrat-Charvillon, and G. Zémor. ZesT: an all-purpose hash function based on Zémor-Tillich. In preparation, 2008.

[204] C. Petit, K. Lauter, and J.-J. Quisquater. Full cryptanalysis of LPS and Morgenstern hash functions. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2008.

[205] C. Petit, K. Lauter, and J.-J. Quisquater. Full cryptanalysis of LPS and Morgenstern hash functions. Cryptology ePrint Archive, Report 2008/173, 2008. `http://eprint.iacr.org/`.

[206] C. Petit, K. E. Lauter, and J.-J. Quisquater. Cayley hashes: A class of efficient graph-based hash functions. Preprint, 2007.

[207] C. Petit, J.-P. Tillich, G. Zémor, and J.-J. Quisquater. Hard and easy components of collision search for the Zémor-Tillich hash function: new attacks and reduced variants with the same security. To appear in CT-RSA 2009, 2008.

[208] C. Petit, N. Veyrat-Charvillon, and J.-J. Quisquater. Efficiency and Pseudo-Randomness of a Variant of Zémor-Tillich Hash Function. In *IEEE International Conference on Electronics, Circuits, and Systems, ICECS2008*, 2008.

[209] R. C.-W. Phan and D. Wagner. Security considerations for incremental hash functions based on pair block chaining. *Computers & Security*, 25(2):131–136, 2006.

[210] A. K. Pizer. Ramanujan graphs and Hecke operators. *Bulletin of the American Mathematical Society*, 23:127–137, 1990.

[211] D. Pointcheval. The composite discrete logarithm and secure authentication. In Imai and Zheng [135], pages 113–128.

[212] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[213] B. Preneel. Hash functions and MAC algorithms based on block ciphers. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, pages 270–282, London, UK, 1997. Springer-Verlag.

[214] B. Preneel. The state of cryptographic hash functions. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, pages 158–182, London, UK, 1999. Springer-Verlag.

[215] B. Preneel. MAC algorithms: State of the art and recent developments. ECRYPT Summer School, May 2008.

[216] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Stinson [256], pages 368–378.

[217] B. Preneel and P. C. van Oorschot. MDx-MAC and building fast MACs from hash functions. *Lecture Notes in Computer Science*, 963:1–14, 1995.

[218] B. Preneel and P. C. van Oorschot. On the security of two MAC algorithms. In *EUROCRYPT*, pages 19–32, 1996.

[219] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search? application to DES (extended summary). In *EUROCRYPT*, pages 429–434, 1989.

[220] J.-J. Quisquater and J.-P. Delescaille. How easy is collision search. new results and applications to DES. In Brassard [55], pages 408–413.

[221] J.-J. Quisquater and M. Joye. Authentication of sequences with the SL_2 hash function: Application to video sequences, 1997.

[222] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[223] V. Rajan. Cryptographic hash functions from expander graphs. Master's thesis, Ecole Polytechnique Fédérale de Lausanne, 2008.

[224] S. Ramanujan. On certain arithmetical functions. In *Transactions of the Cambridge Philosophical Society*, number 22, pages 159–184, 1916.

[225] O. Regev. On the complexity of lattice problems for polynomial approximation factors. In *Proceedings of the LLL+25 conference*, 2007.

[226] O. Reingold. On black-box separations in cryptography. Invited talk at TCC2006., 2006.

[227] R. Rivest. The MD5 message-digest algorithm, 1992.

[228] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[229] M. Rjaško. Properties of cryptographic hash functions. Cryptology ePrint Archive, Report 2008/527, 2008. `http://eprint.iacr.org/`.

[230] P. Rogaway. Bucket hashing and its application to fast message authentication. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 1995.

[231] P. Rogaway. Formalizing human ignorance. In P. Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[232] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[233] A. Russell and H. W. 0002. How to fool an unbounded adversary with a short key. In Knudsen [152], pages 133–148.

[234] M.-J. O. Saarinen. Security of VSH in the real world. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 95–103. Springer, 2006.

[235] B. Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[236] C.-P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9(1):47–62, 1988.

[237] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[238] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Fundamentals of Computation Theory*, pages 68–85, 1991.

[239] C. P. Schnorr, H. H. H. Orner, J. Wolfgang, and G. universitat Frankfurt. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. pages 1–12. Springer Verlag, 1995.

[240] J.-P. Serre. *Linear Representations of Finite Groups*. Springer Verlag, October 1996.

[241] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *23rd annual symposium on Foundations Of Computer Science (Chicago, Ill., 1982)*, pages 145–152. 1982.

[242] A. Shamir. Random graphs in cryptography. Invited talk at Asiacrypt 2006, 2006.

[243] A. Shamir. SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags, 2008.

[244] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.

[245] V. Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inf. Process. Lett.*, 33(5):261–267, 1990.

[246] V. Shoup. A composition theorem for universal one-way hash functions. In *EUROCRYPT*, pages 445–452, 2000.

[247] V. Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[248] J. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986.

[249] D. R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *EUROCRYPT*, pages 334–345, 1998.

[250] D. A. Spielman. Spectral graph theory and its applications. Lecture Notes of Applied Mathematics 500A at Yale University, 2004.

[251] R. Steinwandt, M. Grassl, W. Geiselmann, and T. Beth. Weaknesses in the $SL_2(\mathbb{F}_{2^n})$ hashing scheme. In *Proceedings of Advances in Cryptology - CRYPTO 2000: 20th Annual International Cryptology Conference*, 2000.

[252] M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[253] I. Stewart. *Galois Theory, 3d edition.* CRC Press, 2004.

[254] D. R. Stinson. On the connections between universal hashing, combinatorial designs and error-correcting codes.

[255] D. R. Stinson. Universal hashing and authentication codes. In Feigenbaum [97], pages 74–85.

[256] D. R. Stinson, editor. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer, 1994.

[257] D. R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.

[258] J.-P. Tillich and G. Zémor. Hashing with SL_2. In Desmedt [89], pages 40–49.

[259] J.-P. Tillich and G. Zémor. Collisions for the LPS expander graph hash function. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.

[260] J.-P. Tillich and G. Zémor. Group-theoretic hash functions. In *Proceedings of the First French-Israeli Workshop on Algebraic Coding*, pages 90–110, London, UK, 1993. Springer-Verlag.

[261] P. C. van Oorschot and M. J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 210–218, New York, NY, USA, 1994. ACM.

[262] J. Vélu. Isogénies entre courbes elliptiques. *Communications de l'Académie royale des Sciences de Paris*, 273:238–241, 1971.

[263] D. Wagner. A generalized birthday problem. In Yung [272], pages 288–303.

[264] D. Wagner, editor. *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*. Springer, 2008.

[265] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In Shoup [247], pages 17–36.

[266] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In Shoup [247], pages 1–16.

[267] L. Watanabe, Dai Hitachi. A note on the security proof of Knudsen-Preneel construction of a hash function. NIST SECOND CRYPTOGRAPHIC HASH WORKSHOP, August 2006.

[268] M. Wegman and J. Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22:265—279, 1981.

[269] E. W. Weisstein. Irreducible polynomial. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/IrreduciblePolynomial.html`.

[270] Wikipedia. Linear feedback shift register, October 2008.

[271] R. S. Winternitz. A secure one-way hash function built from DES. In *IEEE Symposium on Security and Privacy*, pages 88–90, 1984.

[272] M. Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

[273] G. Yuval. How to swindle Rabin. *Cryptologia*, 3:187–189, 1979.

[274] G. Zémor. Hash functions and graphs with large girths. In *EUROCRYPT*, pages 508–511, 1991.

[275] G. Zémor. Hash functions and Cayley graphs. *Des. Codes Cryptography*, 4(4):381–394, 1994.

Part IV

Appendices

Appendix A

Publications list

A.1 Cryptographic Hash Functions from Expander Graphs

Title: ZesT: an all-purpose hash function based on Zémor-Tillich
Authors: Christophe Petit, Giacomo de Meulenaer, Jean-Jacques Quisquater, Jean-Pierre Tillich, Nicolas Veyrat-Charvillon and Gilles Zémor
Publication information: Preprint (2009)
Abstract: "Provable" hash functions, the collision resistance of which relies on hard mathematical problems, are very appealing since collision resistance is by far the most important property that a hash function should satisfy. However, provable hash functions tend to be slower than specially-designed hash functions like SHA, and their algebraic structure often implies homomorphic properties and weak behaviors on particular inputs. We introduce the ZesT hash function, a provable hash function that is based on the Zémor-Tillich hash function. ZesT is provably collision and preimage resistant if the balance problem corresponding to Zémor-Tillich is hard. It is currently between 4 and 10 times slower than SHA in software but it has comparable efficiency on FPGA and it admits an ultra-low weight implementation in ASIC. The function computation can be parallelized greatly and efficiently, and its simplicity will certainly allow for a much wider range of implementations and for software-optimized code generation. A careful examination and pseudorandom tests performed with the Dieharder revealed no apparent malleability weakness, which suggests that the function can be used as a general-purpose hash function. Moreover, ZesT can be slightly modified to reach all requirements of the NIST competition. ZesT will really become practical if the hardness of the balance problem corresponding to Zémor-Tillich becomes better established.

Title: Hardware Implementations of a Variant of the Zémor-Tillich Hash Function: Can a Provably Secure Hash Function be very efficient?
Authors: Giacomo de Meulenaer, Christophe Petit and Jean-Jacques Quisquater
Publication information: Submitted (2009)
Abstract: Hash functions are widely used in cryptography, and hardware implementations of hash functions are of interest in a variety of contexts such as speeding up the computations of a network server or providing authentication in small electronic devices as RFID tags. Provably secure hash functions, the security of which relies on the hardness of a mathematical problem, are particularly appealing for security, but they used to be too inefficient in practice. In this paper, we study the efficiency in hardware of ZT', a provably secure hash function based on the Zémor-Tillich hash function. We consider three kinds of implementations targeting a high throughput and a low area in different ways. We first present a high-speed implementation of ZT' on FPGA that is nearly half as efficient as state-of-the-art SHA implementations in terms of throughput per area. We then focus on area reduction and present an ASIC implementation of ZT' with area costs much smaller than SHA-1 and even than SQUASH, which was specially designed for low-cost RFID tags. Between these two extreme implementations, we show that the throughput and area can be traded with a lot of flexibility. Finally, we show that the inherent parallelism of ZT' makes it particularly suitable for applications requiring high speed hashing of very long messages. Our work, together with existing reasonably efficient software implementations, shows that this variant of the Zémor-Tillich hash function is in fact very practical for a wide range of applications, while having a security related to the hardness of a mathematical problem and significant additional advantages such as scalability and parallelism.

Title: Hard and easy Components of Collision Search in the Zémor-Tillich Hash Function: New Instances and Reduced Variants with equivalent Security
Authors: Christophe Petit, Jean-Jacques Quisquater, Jean-Pierre Tillich and Gilles Zémor
Publication information: To appear in the proceedings of CT-RSA 2009
Abstract: The Zémor-Tillich hash function has remained unbroken since its introduction at CRYPTO'94. We present the first generic collision and preimage attacks against this function, in the sense that the attacks work for any parameters of the function. Their complexity is the cubic root of the birthday bound; for the parameters initially suggested by Tillich and Zémor they are very close to being practical. Our attacks exploit a separation of

the collision problem into an easy and a hard component. We subsequently introduce two variants of the Zémor-Tillich hash function with essentially the same collision resistance but reduced outputs of 2n and n bits instead of 3n bits. Our second variant keeps only the hard component of the collision problem; for well-chosen parameters the best collision attack on it is the birthday attack.

Title: Full Cryptanalysis of LPS and Morgenstern Hash Functions
Authors: Christophe Petit, Kristin Lauter, and Jean-Jacques Quisquater
Publication information: SCN 2008 - Proceedings of the Sixth Conference on Security and Cryptography for Networks
Abstract: Collisions in the LPS cryptographic hash function of Charles, Goren and Lauter have been found by Zémor and Tillich [CRYPTO'94], but it was not clear whether computing preimages was also easy for this hash function. We present a probabilistic polynomial time algorithm solving this problem. Subsequently, we study the Morgenstern hash, an interesting variant of LPS hash, and break this function as well. Our attacks build upon the ideas of Zémor and Tillich but are not straightforward extensions of it. Finally, we discuss fixes for the Morgenstern hash function and other applications of our results.

Title: Efficiency and Pseudo-Randomness of a Variant of Zémor-Tillich Hash Function
Authors: Christophe Petit, Nicolas Veyrat-Charvillon, and Jean-Jacques Quisquater
Publication information: WIC'2008 - Symposium on Information Theory and Communication in the Bénélux *and* ISECS'2008 - Proceedings of the 15th IEEE International Conference on Electronics, Circuits and Systems (invited paper)
Abstract: Recent breakthroughs concerning the current standard SHA-1 prompted NIST to launch a competition for a new secure hash algorithm. *Provably secure* hash functions (in the sense that their security relates to the hardness of some mathematical problems) are particularly interesting from a theoretical point of view but are often much slower than heuristic functions like SHA. In this paper, we consider a variant of ZT hash, a provably secure hash function designed by Zémor and Tillich proposed in 1994.Despite some attack proposals, its security has not been fundamentally challenged to this day. Our function is twice as fast as ZT hash and has enhanced security properties. We propose optimized parameters and algorithms to increase the speed of both hash functions. This makes our function one of the most efficient "provably secure" hash functions to this day. Finally, we show that

our hash function successfully passes most pseudo-randomness tests in the Dieharder suite.

Title: Cayley Hashes: A Class of Efficient Graph-based Hash Functions
Authors: Christophe Petit, Kristin Lauter, and Jean-Jacques Quisquater
Publication information: Preprint (2007)
Abstract: Hash functions are widely used in cryptography. Recent breakthroughs against the standard SHA-1 prompted NIST to launch a competition for a new secure hash algorithm, SHA-3. *Provably secure* hash functions, that is functions whose security reduces to a simply-stated, supposedly hard mathematical problem, are widely believed to be much too slow for the NIST competition. In this paper, we discuss Cayley hashes, a class of efficient and provably secure hash functions constructed from the Cayley graphs of (projective) linear groups. We review two existing constructions, the ZT and LPS hash functions, and put a new one forward, the Morgenstern hash function. We show that Cayley hashes are "provable" *and* efficient: on one hand, their security reduces to a representation problem in (projective) linear groups; on the other hand, they are only 5 times slower than SHA-2 in FPGA hardware, and about 400 times slower in software (in our future implementations, many optimizations currently under investigation are expected to decrease these gaps even more). Last but not least, Cayley hash computation can be easily parallelized. We believe their nice properties as well as their elegant design make Cayley hashes very interesting hash functions.

A.2 Physical security

Title: Fault Attacks on Public Key Elements: Application to DLP based Schemes
Authors: Chong Hee Kim, Philippe Bulens, Christophe Petit, and Jean-Jacques Quisquater
Publication information: Proceedings of the Fifth European PKI Workshop - EUROPKI 2008
Abstract: Many cryptosystems suffer from fault attacks when implemented in physical devices such as smart cards. Fault attacks on secret key elements have successfully targeted many protocols relying on the Elliptic Curve Discrete Logarithm Problem (ECDLP), the Integer Factorization Problem (IFP) or the Discrete Logarithm Problem (DLP). More recently, faults attacks have also been designed against the public key elements of ECDLP and IFP-based schemes. In this paper, we present the first fault attacks on the public key

elements of DSA and ElGamal, two DLP-based signature schemes. Our attacks fully recover a 160-bit DSA secret key and a 1024-bit ElGamal secret key with $\sim 4 \cdot 10^7$ and $\sim 3 \cdot 10^6$ faulty signatures respectively. Such figures might suggest that DLP-based schemes are less prone to fault attacks than ECDLP- and IFP-based schemes. However, the integrity of public keys should always be checked in order to thwart such attacks since improvements may reduce the required amount of faulty signatures in the near future.

Title: A Block Cipher based Pseudo Random Number Generator Secure Against Side-Channel Key Recovery
Authors: Christophe Petit, François-Xavier Standaert, Olivier Pereira, Tal G. Malkin, Moti Yung
Publication information: ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security
Abstract: We study the security of a block cipher-based pseudorandom number generator (PRNG), both in the black box world and in the physical world, separately. We first show that the construction is a secure PRNG in the black box world, relying on standard computational assumptions. Then, we demonstrate its security against a Bayesian side-channel key recovery adversary. As a main result, we show that our construction guarantees that the success rate of the adversary does not increase with the number of physical observations, but in a limited and controlled way. Besides, we observe that, under common assumptions on side-channel attack strategies, increasing the security parameter (typically the block cipher key size) by a polynomial factor involves an increase of a side-channel attack complexity by an exponential factor, as usually expected for secure cryptographic primitives. Therefore, we believe this work provides a first interesting example of the way the algorithmic design of a cryptographic scheme influences its side-channel resistance.

Appendix B

Some mathematics and computer science background

B.1 Computational complexity theory

The security of cryptographic algorithms most often relies on the *hardness* of some computational problems. The relative hardness of these problems is characterized by computational complexity theory, a branch of the theory of computation initiated by Alan Turing.

Complexity theory characterizes the amount of resources needed to solve computational problems asymptotically (when the size of the problem increases) and no matter of the actual computing device that is used. Let $f(x)$ and $g(x)$ be two real functions. We say that $f(x)$ is $O(g(x))$ or that $g(x)$ is $\Omega(f(x))$ if there exists a positive real number M and a real number x_0 such that $x > x_0 \Rightarrow |f(x)| \leq M|g(x)|$. We say that $f(x)$ is $\theta(g(x))$ if it is $O(g(x))$ and $\Omega(f(x))$. We say that $f(x)$ is $o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$.

Since the work of Alan Turing, algorithms are modeled by Turing machines. In this thesis, we use a *non-uniform* computation model, meaning that a different Turing machine can be used for each input size (we refer to [137] for a formal definition of this model). We say that an algorithm runs *in probabilistic polynomial time* (PPT) if its expected running time on inputs of size x is $O(x^n)$ for some n.

A fundamental distinction in complexity theory is made between the algorithms running in polynomial time and other algorithms. A *decisional problem* is a problem which answer is 0 or 1. A *complexity class* is a class of problems that can be solved with the same computational resources. The complexity class P is the class of decisional problems that can be solved in deterministic polynomial time. The complexity class NP is the class of decisional problems whose positive answer can be verified in polynomial time given the right information. A very important question in complexity theory is whether P=NP, that is whether all decisional problems that can be verified in polynomial time can also be solved in polynomial time. A problem is NP-complete if it is in the class NP and if its belonging to P is equivalent to the belonging to P of any problem of NP. Some known NP-complete problems are the SAT problem, the knapsack problem, the subset sum problem or the Hamiltonian path problem. Similarly, a problem is in PSPACE if it can be solved with a polynomial amount of memory, and it is PSPACE-complete if its belonging to P is equivalent to the belonging to P of any PSPACE problem. We refer to textbooks (for example [29]) for rigorous definitions of these concepts.

Despite its usefulness at characterizing the complexity of problems, computational complexity theory is irrelevant for cryptography. First, it only describes *asymptotic* hardness, while for example knapsack problems might very well be easy for parameters of practical sizes and become difficult only for very large parameters. Second, it only characterizes *worst-case* hardness, meaning that NP-complete problems may be hard only for some very specific parameters and easy in average, while hardness on average would be required for cryptography. Finally, we point out that the complexity of most used "hard problems" in cryptography, the discrete logarithm, the factorization and the elliptic curve discrete logarithm problems, is only known based on existing attacks and not in the sense of computational complexity theory.

In a sense, cryptography needs its own complexity theory based on average complexity rather than worst-case complexity. Many results in the literature prove the existence or non-existence of certain cryptographic primitives based solely on the existence of other primitives. Impagliazzo [136] summarized these results into five possible "worlds" somehow reminiscent of complexity classes in computational complexity theory. (A synthetic view of these worlds can be found in [226].) In the world MiniCrypt, private key encryption, pseudorandom generators, one-way functions and digital signatures exist since each of these primitives exists if one of them exists. The world Cryptomania additionally contains trapdoor permutations, hence public key encryption, oblivious transfer and key agreement protocols. We refer to [136, 226] and the pointers provided there for further information.

B.2 Matrix theory

We refer to standard textbooks (for example [160]) for elementary notions of matrix theory, including addition and multiplication of matrices; determinant, inverse and rank of a matrix; eigenvalues and eigenvectors; singular value decomposition; matrix norms and the equivalence of matrix norms; symmetric and normal matrices; positive matrices and the Perron-Frobenius theory.

B.3 Groups and fields

A *group* $(G, \oplus)$ is made of a set G and an operation $\oplus : G \times G \to G$ on the elements of G, such that

1. **Associativity:** for all $a, b, c \in G$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$;
2. **Identity:** there exists an identity element e in G, such that for all $a \in G$, $a \oplus e = a = e \oplus a$;
3. **Inverse element:** for all $a \in G$, there exists $b \in G$ such that $a \oplus b = b \oplus a = e$, where e is the identity element.

A group $(G, \oplus)$ is sometimes simply written G when the group operation is clear from the context. A group G is *finite* if G has only a finite number of elements. The *order* of a finite group is the number of its elements. A group is called *Abelian* or *commutative* if for all $a, b \in G$, $a \oplus b = b \oplus a$; otherwise it is called *non-Abelian* or *non-commutative.* A *subgroup* H of a group G is a subset of G that is also a group for the same operation law.

Given a set of elements $S = \{g_1, g_2, ...g_k\} \subset G$, the *subgroup generated by* S is the subgroup made of any element of the form $g_{e_1} \oplus g_{e_2} \oplus ... \oplus g_{e_n}$ for any integer n and any elements $g_{e_i} \in S \cup S^{-1}$ where S^{-1} is the set containing the inverses of the elements of S. The *rank* of a group G is the minimal number of elements needed to generate it. Groups of rank 1 are called *cyclic* groups and are Abelian. The *order* of an element $g \in G$ is the order of the subgroup it generates.

A subgroup H of a group G defines *left cosets* and right cosets in G as follows: for any element $g \in G$, the left coset of H containing g is $gH := \{g \oplus h | h \in H\}$ (the right coset is defined similarly). A subgroup H is called *normal* if for all $g \in G$, $gH = Hg$. The *quotient* group G/H of a group G by one of its normal subgroups H is a group that results from identifying two elements $g_1, g_2 \in G$ iff $g_1 = g_2 \oplus h$ for some $h \in H$. A *simple* group is a group which is not the *trivial group* (the group made of only the identity element) and whose only normal subgroups are the trivial group and the group itself.

A *group homomorphism* is a map between two groups that preserves the group structure: if $(G, \oplus)$ and $(H, \odot)$ are two groups, a group homomorphism $\varphi : G \to H$ satisfies $\varphi(g_1 \oplus g_2) = \varphi(g_1) \odot \varphi(g_2)$ for all $g_1, g_2 \in G$. A *character*

χ of a cyclic group G is a group homomorphism $\chi : G \to \mathbb{C}$. A *linear representation* of a group G is a group homomorphism $G \to GL(n, \mathbb{C})$ for some integer $n \geq 1$.

A *field* $(K, +, *)$ is a set K together with two operations $+ : K \times K \to K$ and $* : K \times K \to K$ acting on it, such that

1. $(K, +)$ is a group with identity element written 0.

2. $(K^*, *)$ is a group with identity element written 1, where $K^* := K \backslash \{0\}$.

3. **Distributivity:** for any $a, b, c \in K$, $a * (b + c) = (a * b) + (a * c)$.

The operations $+$ and $*$ are respectively called addition and multiplication laws and the groups $(K, +)$ and $(K^*, *)$ are respectively called additive and multiplicative groups of the field K. A *finite* field is a field whose number of elements is finite. An isomorphism between two fields K_1 and K_2 is a bijective map that is a group homomorphism for both the additive and the multiplicative groups.

In this thesis, we mainly work with finite groups and fields. For each prime p, the set of "integers modulo p" (that results from identifying two integers z_1, z_2 if and only if $z_1 = z_2 + kp$ for some integer k) is a finite field denoted $\mathbb{F}_p$ for the usual addition and multiplication operations. Both its additive and multiplicative groups are cyclic groups.

A monic *irreducible* polynomial $P(X)$ over a field K is a polynomial with coefficients in K whose coefficient of higher degree is e (the neutral element of K) and that cannot be factored, meaning there do not exist polynomials $Q(X), R(X)$ such that $P(X) = Q(X)R(X)$ and $\deg(Q), \deg(R) < \deg(P)$. For each prime p and monic irreducible polynomial $P_n(X)$ of degree n over $\mathbb{F}_p$, the set of "polynomials over $\mathbb{F}_p$ modulo $P_n(X)$" is a finite field denoted $\mathbb{F}_{p^n}$ for the usual addition and multiplication operations on polynomials. This field is called an *extension field* of $\mathbb{F}_p$. Both its additive and multiplicative groups are cyclic groups. Actually, any finite field is isomorphic to a field $\mathbb{F}_{p^n}$ for some p and n. The *characteristic* of a field $\mathbb{F}_{p^n}$ is p.

The most important groups for cryptography are the multiplicative groups of finite fields $\mathbb{F}_p$ and $\mathbb{F}_{2^n}$, and the group of points on some elliptic curves (see Section B.5) because the discrete logarithm problem is believed to be hard in these groups. In this thesis, we also work with the general linear groups and special linear groups $GL(2, K)$ and $SL(2, K)$ that are 2×2 matrices with coefficients in K that in the case of $SL(2, K)$ have a determinant equal to the identity element.

We refer to [166] for a very complete description of finite groups. The theory of finite fields can be found in any textbooks including [253]. Representation theory for finite groups is explained in [240].

B.4 Quaternion algebras

An *algebra* A over a field K is a vector space over K that has a multiplicative law $\otimes : V \times V \to V$ that is associative and distributive: for all $x, y, z \in A$ and all $a \in K$, it satisfies $(x \otimes y) \otimes z = x \otimes (y \otimes z)$, $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and $a \cdot (x \otimes y) = (a \cdot x) \otimes y = x \otimes (a \cdot y)$. In the following we write all additive symbols by $+$ and we omit all multiplicative symbols. A *division algebra* is an algebra with a division operation, meaning that there is a neutral element 1 and any non-zero $x \in A$ has an inverse x^{-1} such that $xx^{-1} = x^{-1}x = 1$.

A *quaternion algebra* is an algebra of dimension 4 of a particular kind. When the characteristic of K is not 2, a quaternion algebra has a basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$ such that $\mathbf{i}^2 = a$, $\mathbf{j}^2 = b$, $\mathbf{k} = \mathbf{ij} = -\mathbf{ji}$ for some $a, b \in K^*$. When the characteristic of K is 2 the basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$ satisfies $\mathbf{i}^2 + \mathbf{i} = a$, $\mathbf{j}^2 = b$, $\mathbf{k} = \mathbf{ij} = \mathbf{ji} + \mathbf{j}$ for some $a, b \in K^*$. The elements of a quaternion algebra are called *quaternions*. The *conjugate* of any quaternion $q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \in A$ is the quaternion $\bar{q} := a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$ when the characteristic is not 2, and $\bar{q} := a + b(\mathbf{i} + 1) + c\mathbf{j} + d\mathbf{k}$ otherwise. The *trace* of a quaternion $q \in A$ is a field element $t \in K$ defined by $t = \mathrm{Tr}(q) := q + \bar{q}$. The *norm* of a quaternion $q \in A$ is a field element $n \in K$ defined by $n = N(q) := q\bar{q} = \bar{q}q$.

A quaternion algebra is isomorphic either to a division algebra or to the 2×2 matrices. Let A be a quaternion algebra over $\mathbb{Q}$ and let v be a place of $\mathbb{Q}$ with completion $\mathbb{Q}_v$ (that is, $\mathbb{Q}_v$ is either the real numbers if $v = \infty$ or the p-adic numbers if $v = p$ for some prime p). Then A is *split* or *unramified* at v if A becomes isomorphic to the 2×2 matrices over $\mathbb{Q}_v$, and A is *non-split* or *ramified* at v if A becomes isomorphic to the division quaternion algebra over $\mathbb{Q}_v$.

Let A be an algebra finitely generated over $\mathbb{Q}$. An *order* O of A is a subalgebra of A that is finitely generated over $\mathbb{Z}$ and which tensor product with $\mathbb{Q}$ is A. A *maximal order* in A is an order of A that is not contained in any larger order.

We refer to the standard textbook [161] for more details on algebra.

B.5 Elliptic curves

An elliptic curve over a field K is a set of points $(x, y) \in K^2$ satisfying an equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

of the equation) together with a "point at infinity" O with no singular point (no point being a "double solution"); we will write this set $E(K)$ or simply E. When the characteristic of K is neither 2 nor 3, the curve can be given in Weierstrass form

$$E : y^2 = x^3 + a_4x + a_6$$

by changing coordinates. The set of points of an elliptic curve can be given an Abelian group structure with O as neutral element; additions formulae are given in [248], Section 3.2. The l-torsion of an elliptic curve is the subgroup made of the points of order dividing l in any sufficiently large extension of K.

Elliptic curves as groups have become very important in cryptography since their introduction in 1985 by Koblitz [154] and Miller [186]. Under some conditions, the discrete logarithm problem on elliptic curves is believed to be much harder than the discrete logarithm problem on the multiplicative group of finite fields of equal size.

Two elliptic curves E and E' are isomorphic if there exists a change of coordinates mapping the points of E to the points of E'; isomorphic curves are often thought of as a single curve represented by two different equations. Isomorphic elliptic curves have a same j-invariant, defined for Weierstrass equations by $j(E) := 1728\frac{4a_4^3}{4a_4^3+27a_6^2}$.

Given two elliptic curves E, E' defined over the same field, a homomorphism from E to E' is a rational map preserving the group addition. An isogeny from E to E' is a non-zero homomorphism; its degree is the cardinality of its kernel. An isogeny from E to itself is called an endomorphism. The set of endomorphisms of an elliptic curve is a ring and is isomorph either to $\mathbb{Z}$, to an order in a quadratic number field or to an order in a quaternion algebra. An isogeny of degree 1 is called an automorphism.

When K is a finite field $\mathbb{F}_q$ of characteristic p, an elliptic curve over $\mathbb{F}_q$ is supersingular if for every finite extension $\mathbb{F}_{q^r}$, the curve $E(\mathbb{F}_{q^r})$ has no point of order p. The j-invariants of supersingular elliptic curves are called supersingular j-invariants. The endomorphism ring of a supersingular elliptic curve is an order in a quaternion algebra.

The main reference on elliptic curves is Siverman's book [248]. For elliptic curve cryptography, a good reference is Blake et al.'s book [47]. Beginners will find easier to start with Joye's master thesis [147].

Appendix C

LPS and Morgenstern computation for $l = 5$ and $q = 2$

C.1 LPS hash function with $l = 5$

If $l = 5$, the 6 graph generators for LPS hash function are

$$s_{\pm 1} = \begin{pmatrix} 1 \pm 2\mathbf{i} & 0 \\ 0 & 1 \mp 2\mathbf{i} \end{pmatrix}, \quad s_{\pm 2} = \begin{pmatrix} 1 & \pm 2 \\ \mp 2 & 1 \end{pmatrix}, \quad \text{and } s_{\pm 3} = \begin{pmatrix} 1 & \pm 2\mathbf{i} \\ \pm 2\mathbf{i} & 1 \end{pmatrix}.$$

Let $M = \begin{pmatrix} a_0 + a_1\mathbf{i} & b_0 + b_1\mathbf{i} \\ c_0 + c_1\mathbf{i} & d_0 + d_1\mathbf{i} \end{pmatrix}$ with $a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1 \in \mathbb{F}_p$. The following equalities result from $\mathbf{i}^2 = -1$.

$$\begin{aligned}
Ms_{\pm 1} &= \begin{pmatrix} (a_0 \mp 2a_1) + (a_1 \mp 2a_0)\mathbf{i} & (b_0 \pm 2b_1) + (b_1 \mp 2b_0)\mathbf{i} \\ (c_0 \mp 2c_1) + (c_1 \mp 2c_0)\mathbf{i} & (d_0 \pm 2d_1) + (d_1 \mp 2d_0)\mathbf{i} \end{pmatrix}, \\
Ms_{\pm 2} &= \begin{pmatrix} (a_0 \mp 2b_0) + (a_1 \mp 2b_1)\mathbf{i} & (b_0 \pm 2a_0) + (b_1 \pm 2a_1)\mathbf{i} \\ (c_0 \mp 2d_0) + (c_1 \mp 2d_1)\mathbf{i} & (d_0 \pm 2c_0) + (d_1 \pm 2c_1)\mathbf{i} \end{pmatrix}, \\
Ms_{\pm 3} &= \begin{pmatrix} (a_0 \mp 2b_1) + (a_1 \pm 2b_0)\mathbf{i} & (\mp 2a_1 + b_0) + (\pm 2a_0 + b_1)\mathbf{i} \\ (c_0 \mp 2d_1) + (c_1 \pm 2d_0)\mathbf{i} & (\mp 2c_1 + d_0) + (\pm 2c_0 + d_1)\mathbf{i} \end{pmatrix}.
\end{aligned}$$

Computing each of these formulae only requires 8 multiplications by 2 and 8 additions in $\mathbb{F}_p$. A multiplication by 2 amounts to shifting the bits, testing the left-most bit and adding p if this bit is 1, that is every two times in mean. An addition in $\mathbb{F}_p$ amounts to an integer addition plus another addition by $-p$ if the left-most bit of the result is 1, that is every two times in mean. As $\log_2 5$ bits are processed by step, the cost per bit is therefore about 7.75 additions and 3.45 one-bit shifts per bit of message.

C.2 Morgenstern hash function with $q = 2$

When $q = 2$, the 3 graph generators of Morgenstern hash are

$$s_0 = \begin{pmatrix} 1 & 1 \\ X & 1 \end{pmatrix}, \quad s_1 = \begin{pmatrix} 1 & \mathbf{i} \\ X(1+\mathbf{i}) & 1 \end{pmatrix}, \quad \text{and } s_2 = \begin{pmatrix} 1 & 1+\mathbf{i} \\ X\mathbf{i} & 1 \end{pmatrix}.$$

Let $M = \begin{pmatrix} a_0 + a_1\mathbf{i} & b_0 + b_1\mathbf{i} \\ c_0 + c_1\mathbf{i} & d_0 + d_1\mathbf{i} \end{pmatrix}$ with $a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1 \in \mathbb{F}_{2^n}$. The following equalities result from $\mathbf{i}^2 + \mathbf{i} + 1 = 0$.

$$\begin{aligned}
Ms_0 &= \begin{pmatrix} (a_0 + b_0X) + (a_1 + b_1X)\mathbf{i} & (a_0 + b_0) + (a_1 + b_1)\mathbf{i} \\ (c_0 + d_0X) + (c_1 + d_1X)\mathbf{i} & (c_0 + d_0) + (c_1 + d_1)\mathbf{i} \end{pmatrix}, \\
Ms_1 &= \begin{pmatrix} (a_0 + b_0X + b_1X) + (a_1 + b_0X)\mathbf{i} & (a_1 + b_0) + (a_0 + a_1 + b_1)\mathbf{i} \\ (c_0 + d_0X + d_1X) + (c_1 + d_0X)\mathbf{i} & (c_1 + d_0) + (c_0 + c_1 + d_1)\mathbf{i} \end{pmatrix}, \\
Ms_2 &= \begin{pmatrix} (a_0 + b_1X) + (a_1 + b_0X + b_1X)\mathbf{i} & (a_0 + a_1 + b_0) + (a_0 + b_1)\mathbf{i} \\ (c_0 + d_1X) + (c_1 + d_0X + d_1X)\mathbf{i} & (c_0 + c_1 + d_0) + (c_0 + d_1)\mathbf{i} \end{pmatrix}.
\end{aligned}$$

Computing the last two formulae only requires 4 multiplications by X and 12 additions in $\mathbb{F}_{2^n}$, and in the first formula the number of additions is even reduced to 8. A multiplication by X amounts to shifting the bits, testing the left-most bit and xoring with $P_n(X)$ if this bit is 1, that is every two times in mean. In mean, the cost per bit of message is about 4 one-bit shifts and 12.67 XORs.

Appendix D

Generation of ZesT's constants

Here is the C++ code that we have used to generate the constants $P_n(X)$ and $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right)$ in the fixed-key version of ZesT.

```
#include <time.h>
#include <fstream>
#include <iomanip>
#include <NTL/GF2.h>
#include <NTL/vec_GF2.h>
#include <NTL/GF2XFactoring.h>
#include <NTL/mat_GF2.h>

NTL_CLIENT


void getparam(GF2X& pol, vec_GF2& a0,
                        vec_GF2& b0, int n, uint16_t init);


int main(int argc, char * argv[])
{
    // n = 223,251,383,509
    int n ;
    sscanf(argv[1], "%d", &n);
    cout<<"n is "<<n<<endl;

    // for init we start from the binary representation of pi
    // 11.001001000011111101101010100010
    uint16_t init = 0xc90f ;
    cout<<"init is set to "<<init<<endl;

    GF2X pol = GF2X() ;
    vec_GF2 a0 = vec_GF2(INIT_SIZE, n);
    vec_GF2 b0 = vec_GF2(INIT_SIZE, n);

    // get the parameters
    getparam(pol,a0,b0,n, init);
}

// Returns a polynomial of the form
// x^n+ 1 + pol(LFSR(seed))
// and two constant a0 and b0 'pseudorandom'
void getparam(GF2X& pol, vec_GF2& a0,vec_GF2& b0,
                            int n, uint16_t init)
{
    // Form the first polynomial

    SetCoeff(pol,n,1);
    SetCoeff(pol,0,1);

    // The lfsr we choose is the Fibonacci lfsr defined by
    // the polynomial x^16 + x^14 +x^13 +x^11 +1
    int i,bit = 0;
    uint16_t reg = init ;
    for (i =1;i<n;i++)
    {
        bit = (reg ^ (reg >> 2)
                ^ (reg >> 3) ^ (reg >> 5))
               & 1;
        reg = (reg >> 1) | (bit << 15);
        SetCoeff(pol,i,bit);
    }


    // check irreducibility
    int flag = IterIrredTest(pol) ;

    // update polynomial with the LFSR
    // until you get an irreducible polynomial
    while(!flag)
    {
        i++;
        cout<<"i is "<<i<<endl<<endl;
        // modify the polynomial
        bit = (reg & 0x0001) ^ ((reg & 0x0004) >> 2)
                ^ ((reg & 0x0008) >> 3) ^ ((reg & 0x0020) >> 5);
        reg = (reg >> 1) | (bit << 15);
        SetCoeff(pol,0,0);
        RightShift(pol,pol,1);
        SetCoeff(pol,0,1);
        SetCoeff(pol,n-1,bit);
        SetCoeff(pol,n,1);

        // check irreducibility
        flag = IterIrredTest(pol) ;
    }

    // Compute the constants a0 and b0
    for(i=0;i<n;i++)
    {
        bit = (reg & 0x0001) ^ ((reg & 0x0004) >> 2)
                ^ ((reg & 0x0008) >> 3) ^ ((reg & 0x0020) >> 5);
        reg = (reg >> 1) | (bit << 15);
        b0.put(i,bit);
    }
    for(i=0;i<n;i++)
    {
        bit = (reg & 0x0001) ^ ((reg & 0x0004) >> 2)
                ^ ((reg & 0x0008) >> 3) ^ ((reg & 0x0020) >> 5);
        reg = (reg >> 1) | (bit << 15);
        a0.put(i,bit);
    }
}
```

Appendix E

Examples for our algorithms of Chapter 6

E.1 Toy example of the preimage-finding (path-finding) algorithm in the LPS graph

As an example of our preimage algorithm, we now give a second preimage for the message $m =$"This is not for NIST", when the parameters are $p = 1125899906842769$ and $l = 5$. The ASCII code for m is 01010100 01101000 01101001 01110011 00100000 01101001 01110011 00100000 01101110 01101111 01110100 00100000 01100110 01101111 01110010 00100000 01001110 01001001 01010011 01010100 which in base 5 gives 30232314430000323121040012440301 342104032442012221213343131044243 2021. We start at the identity, with g_0 the identity and the starting edge (s_1^-1, I). We identify the six graph generators

$$s_{\pm 1} = \begin{pmatrix} 1 \pm 2\mathbf{i} & 0 \\ 0 & 1 \mp 2\mathbf{i} \end{pmatrix}, \quad s_{\pm 2} = \begin{pmatrix} 1 & \pm 2 \\ \mp 2 & 1 \end{pmatrix} \quad s_{\pm 3} = \begin{pmatrix} 1 & 2\mathbf{i} \\ 2\mathbf{i} & 1 \end{pmatrix}$$

with their indices. The function π we choose is given in figure E.1. The hash value obtained is

$$M = \begin{pmatrix} 1113908155375639 & 815055784352014 \\ 485525153198538 & 30164330826615 \end{pmatrix}.$$

We apply our path-finding algorithm on M. First, we get a matrix decomposition as in Section 6.2. After 11 trials, the resulting λ, α, ω, β_1 and

Table E.1: Function π: the table gives the index of the next matrix for a given current matrix and a given base 5 digit.

	-3	-2	-1	1	2	3
0	-3	3	2	1	-1	-2
1	2	-3	3	2	1	-1
2	-1	-2	-3	3	2	1
3	1	-1	-2	-3	3	2
4	2	1	-1	-2	-3	3

β_2 values are

$$\begin{aligned}\lambda &= 1051846637406052\\ \alpha &= 698130975272599\\ \omega &= 846326642296745\\ \beta_1 &= 150389273084944\\ \beta_2 &= 480539407839455.\end{aligned}$$

Then we factorize

$$\begin{aligned}M_\alpha &:= \begin{pmatrix} 1 & 0\\ 0 & \alpha\end{pmatrix}\\ &= = \begin{pmatrix} \begin{array}{c}349065487636300\\ +\mathbf{i}795285597612250\end{array} & 0\\ 0 & \begin{array}{c}349065487636300\\ -\mathbf{i}795285597612250\end{array}\end{pmatrix}.\end{aligned}$$

We choose $k = 48$, resulting in $\lambda =$222458048101540 and $m =$11210387681441600668869823936886993015607319565640625. After 234 random trials for x, we finally get $x = 523712450310834$, $w = 207632734870715$, and $n = 4.2489205976128525372183128649803320961$. The Euclidean algorithm gives us the solution $y =$2782001231666122912, $z =$1489057773063985790. So the lift of M_α is $\widetilde{M_\alpha} =$

$$\left(\begin{array}{c|c} \begin{array}{c}311426103887630914544037511835\\ +\mathbf{i}766565480745454184887163124346\end{array} & \begin{array}{c}3132254927569356406015273012423328\\ +\mathbf{i}1676530007976242663293697980252510\end{array}\\ \hline \begin{array}{c}-3132254927569356406015273012423328\\ +\mathbf{i}1676530007976242663293697980252510\end{array} & \begin{array}{c}311426103887630914544037511835\\ -\mathbf{i}766565480745454184887163124346\end{array}\end{array}\right).$$

We multiply $\widetilde{M_\alpha}$ by each of the lifts of the graph generators. Since $\widetilde{M_\alpha}\widetilde{s}_3$ is divisible by $l = 5$, $\widetilde{s}_{-3}$ is the last (right-hand) factor of $\widetilde{M_\alpha}$. After $2k$ steps, we get the whole factorization of $\widetilde{M_\alpha}$, which we translate into a factorization of M_α whose indices are 3 -1 2 2 3 1 1 3 1 3 3 3 2 2 3 -1 2 1 1 -3 1 1 1 3 -1 2 -3 2 3 1 -2 -2 -2 1 2 1 1 -3 2 1 2 1 -2 3 -1 3 2 -3 -2 3 1 -2 3 3 2 -3 -1 2 2 2 -1 -3 -1 -3 2 3 1 2 -3 -1 3 2 2 1 3 -2 -3 1 3 -2 -1 -2 3 1 3 2 1 -2 -1 -1 -3 2 1 1 -2 -3. We get the factorizations of M_ω, M_{β_1} and M_{β_2} the same way. Finally, we put all the pieces of information together and get the sequence -3 -2 1 1 2 -3 -1 -1 -2 1 2 3 1 3 -2 -1 -2 3 1 -3 -2 3 1 2 2 3 -1 -3 2 1 3 2 - 3 -1 -3 -1 2 2 2 -1 -3 2 3 3 -2 1 3 -2 -3 2 3 -1 3 -2 1 2 1 2 -3 1 1 2 1 -2 -2 - 2 1 3 2 -3 2 -1 3 1 1 1 -3 1 1 2 -1 3 2 2 3 3 3 1 3 1 1 3 2 2 -1 3 2 3 -1 -3 -2 -2 1 3 2 -3 -3 2 3 -2 -3 -3 -2 -1 -2 3 1 1 2 3 2 1 1 -3 1 2 2 1 -2 1 1 2 -3 -2 3 -2 -3 1 1 3 1 2 1 -3 -1 -3 -3 -1 2 3 1 -3 -3 -1 -1 2 -1 3 -2 1 -3 -3 -1 -2 1 1 -2 -1 -1 3 -2 3 2 2 1 -3 -2 -1 -3 -1 -3 -1 3 2 -1 3 3 -2 -1 -2 1 1 2 2 -3 -1 3 2 2 -3 -2 -3 -1 -3 1 -3 -2 -1 3 1 -2 3 -2 3 2 1 3 -2 -2 -3 -3 -2 -2 3 -2 -2 -3 -1 3 1 3 -1 -3 -3 -3 -3 -2 1 3 3 1 -2 3 -1 -2 1 2 -3 1 -2 -2 1 -2 -2 -1 2 2 2 2 2 1 -3 1 1 2 1 1 3 3 -1 3 3 -2 -1 3 1 2 -1 2 3 -1 2 -3 -2 1 -2 1 1 3 -2 2 -2 -3 -2 -1 3 3 -2 -1 3 2 -3 2 3 -2 1 1 1 1 -2 1 1 2 -1 3 -1 -2 -1 -2 -1 -2 -2 3 2 1 3 - 2 -2 -3 -3 -2 3 3 2 1 1 1 1 3 2 -1 -3 2 -3 -2 -1 -3 1 -2 -2 -2 1 -2 -1 -2 -1 2 - 1 -3 -1 -3 -2 -1 -2 -3 -1 -3 -1 2 2 3 3 -2 -2 -2 -3 -1 -1 -1 2 -3 -1 3 -1 2 -3 - 3 that collides with the original message “This is not for NIST”.

E.2 Second preimage of “This is not for NIST” for LPS hashes with 1024-bit parameters

Now we repeat what we did in Appendix E.1, this time for
$p =$ 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224139329, which has 1024 bits. The sequence we get is 3 1 3 3 2 3 2 3 -2 -1 -2 1 -3 1 2 -1 -3 -2 -3 -1 2 -1 -1 -3 -2 -2 -3 1 2 -3 -3 2 -1 3 2 2 2 -3 -3 1 1 2 -1 3 -2 -2 1 1 -2 -3 2 -3 -3 1 3 1 3 -2 3 2 3 2 3 3 -2 1 1 -2 3 1 3 -2 1 3 1 -3 1 3 3 1 2 2 1 1 1 -2 3 -1 -1 -2 3 2 3 1 3 1 -3 1 1 -3 2 2 1 3 -1 -3 1 2 3 -2 3 2 1 3 2 1 2 3 3 -2 1 2 -1 3 1 3 1 3 -1 3 2 -3 -1 -2 -3 2 -3 -1 -2 3 1 2 -3 -3 -1 -3 -3 -3 -1 -3 -3 1 3 3 3 -2 -1 -2 -2 3 3 1 -2 -2 -2 1 -2 3 1 -3 2 -1 2 3 3 2 1 1 -3 -2 -3 1 -3 -3 -2 -1 -1 -2 -2 -3 -2 -1 2 -3 -3 -3 1 2 -3 -2 -1 2 1 3 3 -2 1 -3 2 3 -2 3 1 -3 -2 -1 3 3 1 3 -1 -2 1 2 -3 -1 3 3 -1 3 -1 -2 3 3 2 2 -3 2 1 2 -1 2 2 -3 -2 -2 1 -3 -3 1 2 2 -3 -1 -3 1 2 -3 2 2 -3 -3 -1 -1 -1 2 1 2 3 -1 -1 3 2 2 -3 -2 1 3 -2 3 -1 3 -2 -2 1 2 1 1 2 -1 -1 -1 3 -1 2 2 2 -1 -3 -3 -3 1 -3 -1 -1 -1 3 3 3 3 -2 -2 3 -1 3 2 -3 1 2 2 -1 -1 3 -2 3 -2 -1 3 -2 3 -2 3 -2 -1 2 1 3 1 -2 3 2 2 3 -1 -1 3 2 2 3 -2 3 -1 -3 2 3 -2 3 1 -3 - 3 -2 -2 1 2 1 3 1 -3 -2 -1 2 2 2 -1 3 3 2 2 2 2 3 3 -1 -1 -1 -3 -2 3 1 2 -1 -2 1 -3 -1 3 -2 -3 1 -3 -2 -2 -1 -1 -1 -1 -3 -3 2 2 1 -3 -3 -2 -3 -1 -3 2 -3 1 -2 1 1 -3 -1 -3 -2 -1 -3 -1 -1 3 3 1 3 -2 3 -2 -3 -1 2 2 3 -1 3 -2 -3 1 3 3 2 2 3 1 1 -3 -1 -3 -1 -2 -2 -1 -3 -3 1 -2 -2 -1 3 3 1 -3 -1 -1 -3 2 3 -1 3 -2 -3 2 -3 -2 1 -3 -1 -1 -2 3 -2 -2 -3 1 -2 -2 -2 3 -1 2 3 -1 2 3 -1 -2 -1 3 -2 1 2 2 3 1 2 -1 2 1 -3 -3 -2 3 3 1 2

```
-3 1 2 1 -2 -1 2 -3 1 2 -3 -3 -1 -1 -3 2 1 3 -1 2 3 3 -2 - 3 1 3 1 -3 -1 -3 2 -1 -3 -3 -1 3 3 -2 -1 -2 1 3 -2 -3 2 1 -2 -1 3 -2 1 2 -1
2 3 1 -2 -3 1 -2 -2 -1 -2 -2 -2 3 2 1 2 -3 -2 -1 -3 1 -3 -2 -2 -1 -2 3 -2 -2 -1 -3 - 3 1 -3 1 3 2 -3 -2 3 3 1 -3 -2 1 3 2 1 3 -2 3 -1
2 -3 -3 2 3 1 -2 -1 -3 -3 1 3 - 2 -3 -2 1 2 1 -3 -1 -2 -3 -3 1 2 3 2 1 1 -3 -3 -2 3 3 3 2 -3 1 -2 1 -2 3 -2 1 3 3 3 2 1 3 1 3 2 1 1
2 -1 -2 1 3 -1 -3 -1 3 3 2 -3 1 3 1 -3 -2 -1 2 2 -3 1 3 3 3 -2 3 3 1 1 3 -2 1 1 1 2 1 -3 -3 -3 -2 -1 -2 -1 3 3 2 -3 1 2 1 -3 1 -3 -3
-2 -2 3 2 2 3 -2 -2 -1 2 -3 -3 2 2 -3 -3 2 -1 -1 -3 2 -3 1 2 1 3 -2 -3 1 -2 1 3 2 -3 -3 -3 -2 -2 1 2 -3 1 2 -1 -1 -3 -2 3 2 2 2 -1 -3
-1 2 1 -2 -2 3 3 2 2 3 -1 -1 -2 - 3 2 3 -2 1 1 -2 -1 -1 -2 1 1 -3 -2 1 1 1 -3 -3 -3 2 2 1 -2 1 -2 -3 1 -3 -1 -2 3 -2 3 1 -3 -1 2 -1 2
-1 -1 3 1 1 -3 -1 -3 2 3 1 -3 -1 2 1 2 -3 1 -3 -2 1 -2 -1 -3 -2 -1 -2 -3 -1 -1 -2 1 -2 1 3 1 3 -2 3 1 3 3 3 3 -1 -2 -3 -1 3 3 -2 -2 -1
-1 3 -1 -2 3 1 -3 1 3 2 3 -2 -2 -3 -2 -2 -1 -2 -3 -1 3 2 1 2 -3 1 2 -3 -2 1 3 3 3 -1 3 2 -1 -2 1 -3 -1 -2 -2 -3 2 1 1 2 2 2 1 2 -1 -2
1 1 2 2 3 -2 1 3 3 -2 -1 -3 2 3 3 -2 -1 2 -3 1 1 1 -2 -2 -3 1 -2 3 -1 3 2 3 3 -2 1 2 1 -3 -2 -2 -2 -1 3 -1 -3 - 3 -2 -1 -3 1 -3 2 -1
2 1 2 1 3 3 3 -1 -1 -2 1 2 -1 -1 3 -2 1 2 3 -1 3 -1 2 -3 -2 -3 1 -2 -3 2 3 3 2 -1 -2 1 -2 -1 -3 -2 -1 -2 3 -1 3 2 -3 -3 -1 3 1 -3 2
-1 -1 2 -3 -3 1 -2 1 -3 -1 2 3 1 1 -2 -3 -2 3 -1 -1 -1 -3 -1 -1 -2 -2 -3 -3 1 -2 -3 2 2 3 3 2 1 3 2 -3 -2 -2 1 2 2 3 -2 3 2 -1 3 3 1
2 3 -2 3 -2 1 -3 1 3 2 1 1 2 2 -1 2 1 2 3 1 -2 1 1 3 -2 1 -2 -3 -2 3 1 -3 -3 1 1 2 1 -2 -2 3 -1 2 -1 -2 -3 -2 1 3 3 1 -3 2 -1 2 -3 2
3 1 1 1 3 2 3 -2 1 -2 1 -2 1 3 -2 -1 2 -3 -1 3 2 -3 1 -2 -1 - 2 -2 -1 2 1 -3 -1 2 -1 -3 2 -1 -1 2 2 -3 -3 1 2 1 3 -1 3 1 2 2 -3 2 -1
2 2 1 1 - 3 -1 -1 -1 3 3 -2 -2 3 -1 -2 -1 3 1 3 3 -2 1 -3 -2 -2 1 -2 1 -2 -1 -3 2 2 -1 -2 -1 -2 -1 -3 -2 -1 -1 2 2 2 2 -3 -2 -2 -2 1 3
1 -2 -3 -1 -1 -2 3 2 3 -1 3 2 -1 -2 3 -1 2 -3 -2 -2 -2 3 -1 3 2 -1 2 2 -1 -3 1 -3 -3 2 1 -2 -1 -2 1 -2 -2 1 2 1 -3 -3 -1 -1 -1 2 -3 -2
-3 1 -2 1 1 1 -2 -2 -1 -1 3 1 2 1 3 -2 -1 -2 3 2 2 2 -3 1 3 1 3 -2 3 -2 -2 -3 -2 -3 -3 2 3 -2 1 1 3 1 -2 -3 -2 -2 1 2 -3 -2 -2 -1 -1
2 -1 -2 3 2 3 -1 -2 3 2 3 -2 -2 -2 1 2 1 -2 3 2 3 -1 -3 -1 -1 3 -1 -2 1 -2 1 2 3 -1 3 1 2 -1 2 1 1 2 3 2 -3 -3 -3 -2 -3 -3 2 -3 -1 3
3 2 2 1 -3 -2 -3 -3 -1 3 -2 -2 -1 3 -1 -3 2 -3 2 2 -3 2 3 3 2 -1 -2 3 3 2 3 2 2 1 -3 2 1 2 2 2 2 -3 -2 1 -3 1 2 3 3 3 3 -2 -2 1 2 -3
-3 2 3 3 2 -1 -1 3 2 2 2 -1 2 -1 -2 3 2 1 1 -2 -2 -2 3 1 1 3 -2 1 2 2 -3 2 1 1 -3 -3 2 2 -1 -2 -3 -1 -3 -3 -3 -1 -1 -2 1 -3 1 -3 -1 -1
-3 -1 -3 1 2 -3 2 1 -3 2 -3 -2 3 2 -3 1 3 1 -2 -3 -3 -1 -3 -2 -3 2 2 2 3 1 1 2 1 1 1 2 2 -1 -2 -2 3 -1 3 -1 -2 3 -1 -3 1 3 3 1 -3 -2
-1 -3 -1 2 3 -2 -1 -2 -1 -3 -1 -3 -1 2 1 3 -2 -2 3 2 1 -2 3 -1 3 1 -2 -1 -1 -3 -2 1 -3 1 -3 -2 -1 3 -1 -2 -1 3 3 2 3 -1 2 1 -3 2 2 -1
-2 -2 -2 3 3 3 -1 -3 -1 -2 3 -1 -1 3 2 1 1 3 3 3 3 1 2 3 2 -3 -1 -1 3 -2 3 2 -1 -2 -3 1 1 -3 -2 -3 -3 1 -3 2 3 -1 -3 2 -3 2 -1 3 1 1
-3 1 - 2 3 2 3 1 3 -1 2 3 -1 2 -3 -2 -2 -1 3 -2 -2 3 2 3 -2 -3 1 3 2 1 3 3 3 -2 -1 2 -1 3 -1 3 -2 1 1 -3 2 3 2 3 -2 -3 -2 -3 -3 -3 -1
-3 -3 -3 -3 2 -1 2 -1 3 -2 1 1 -3 -1 -3 -2 -1 -1 -3 -1 2 2 2 -1 -1 -2 1 2 3 -2 -3 2 2 -1 3 3 -2 -1 -3 -2 -3 1 -3 -1 2 1 2 1 2 2 -1 3 2
2 -3 -3 1 3 -1 2 1 3 2 -3 -3 1 -2 1 1 2 -1 2 2 -1 2 1 1 3 1 -2 1 -2 -1 -3 -1 -2 -1 -3 1 3 3 1 2 3 1 1 -2 -2 1 -2 -2 -1 -3 1 3 1 -3 -3
-3 - 1 -1 3 -1 -1 -3 2 -3 -3 -1 -2 -1 -2 1 1 2 2 2 1 2 2 -1 2 1 3 3 2 3 3 -1 -3 -1 -2 1 3 1 -2 1 -2 1 -3 -3 -2 1 1 -3 2 2 2 3 1 3 2 2
2 -3 2 -1 -3 1 1 -3 1 -2 3 1 2 -3 -3 1 1 -2 -3 2 -1 3 3 -1 -1 -1 -3 1 -3 -1 2 -1 -1 -1 -1 -3 -3 -2 3 1 1 1 -2 - 1 2 3 2 1 -3 1 1 3 -2
-3 2 -3 -2 -2 3 3 1 3 2 -1 3 1 3 -1 -3 -3 1 1 2 -1 -3 -3 1 3 -2 1 -3 2 2 -3 -3 -2 3 2 -3 2 3 3 2 -1 -3 2 1 2 -3 -3 1 -3 2 3 1 -3 -2
-2 -3 -2 -1 -3 2 -1 -2 -3 -2 -1 3 -2 3 -2 -3 -3 2 2 2 1 2 2 -1 -3 -1 -3 -1 -1 2 2 3 1 -2 3 -1 -2 1 3 1 2 1 3 -2 -2 -2 3 2 2 -1 -2 -1
-3 2 -3 -3 -2 3 3 -1 2 2 -1 3 2 - 3 1 3 -2 1 2 2 -1 -2 -2 -2 -2 -3 2 -3 -2 3 -2 3 3 -1 -2 1 -2 3 3 1 -3 2 1 2 2 -1 3 -1 -3 -1 3 -1 -1
3 1 -3 1 -3 2 2 2 -1 -1 2 2 -3 -3 1 -3 1 -3 -1 -3 1 -3 -2 -2 -3 -3 -2 -1 -2 -3 -3 1 -2 -3 -3 -1 -1 -3 -3 -1 -1 2 1 1 -2 -1 2 2 2 1 -3
2 -1 3 1 1 -3 -1 3 -2 3 2 -3 2 1 -2 3 3 -2 -2 3 3 -1 3 3 -1 3 3 1 1 -2 -1 -1 3 -1 -1 2 -3 -1 -3 2 -3 -3 2 1 2 2 -3 1 3 -1 -3 -1 -3 -2
-1 -3 -3 2 1 2 -3 2 -3 1 2 2 -3 -3 -1 2 2 3 1 -3 -1 -2 3 -2 1 1 -2 1 1 -3 -3 2 -3 -1 -2 -2 -2 -1 -1 -3 -3 1 3 1 1 3 2 3 1 1 -3 -3 -1
-3 -2 -3 -3 2 -1 -1 -3 2 -3 -1 -3 1 -2 -1 3 1 1 -3 1 1 3 -2 -2 -3 -1 -2 3 3 3 3 -2 -3 -3 1 2 2 1 1 1 -2 3 3 -2 3 -1 -3 2 -1 2 3 1 3 2
2 1 2 3 1 1 -3 -3 -1 2 -3 -3 -1 -1 2 -1 2 3 2 1 2 2 3 1 1 1 3 -1 -3 -1 3 2 3 -1 -1 -2 3 1 -3 2 1 2 3 -2 1 1 -2 -1 -3 -1 3 2 2 3 -1
-3 1 -3 -2 -2 3 -1 -1 -1 3 3 -2 3 1 -3 -2 3 -2 -1 3 -1 2 1 -3 -3 -3 2 -1 3 3 -2 3 2 1 3 -2 -3 -2 -3 2 2 1 3 1 -2 1 2 3 1 -2 -3 -2 -3
2 2 2 -1 -3 -3 -3 -1 -2 3 3 2 2 -3 2 3 -2 1 -2 -2 3 2 -3 -1 - 3 -1 -1 2 2 2 1 1 2 1 3 3 2 -3 1 1 -3 -3 -2 1 3 3 -2 3 -2 -1 -1 3 3 3
-2 -3 1 1 -3 -3 2 -1 -2 -2 -2 -1 2 1 -3 1 -2 -1 -2 -2 1 3 3 1 2 2 2 2 1 -3 2 2 3 -1 -3 -1 3 -2 1 -3 -2 -1 -1 3 -2 1 -2 1 3 1 -3 1 3 1
3 -2 1 2 3 -1 2 -1 -3 -2 1 3 1 3 1 3 2 -3 1 1 3 1 -3 -2 -3 -3 2 2 1 1 3 -2 3 1 3 3 -1 -1 3 1 -2 -2 1 1 -3 -2 -2 -1 - 2 -3 -3 1 -2 3 2
2 -1 -1 2 -1 3 2 -1 3 1 -3 -2 1 -2 -3 2 -1 -1 2 -3 2 -3 1 3 1 3 -1 2 3 2 2 -3 1 2 2 2 2 2 3 -2 1 2 -3 -2 -1 -2 3 -1 -2 -3 -2 1 2 -3 2
-1 -2 -2 3 3 -2 -3 -3 -3 -2 -3 -2 -2 1 3 -2 3 -1 -3 -1 -1 3 2 2 3 1 -2 3 3 3 -1 -1 3 3 -2 -1 -1 3 3 -1 2 1 2 -1 2 -1 -3 1 2 1 -3 -1 3
```

-1 -2 1 2 -3 -2 -3 -1 -2 1 -2 -2 1 2 -3 2 1 2 -3 -3 -1 -2 -3 -1 2 -1 -2 -3 -3 1 -3 -2 1 2 3 1 2 3 1 -3 -1 2 -3 1 -3 1 1 1 3 -1 -1 -2

-3 2 2 -1 -1 -3 -1 3 -1 -1 3 3 -2 3 -2 -1 -3 -2 -1 -3 -3 1 1 2 2 -3 1 2 1 3 -2 1 -3 2 1 -3 -3 -3 2 3 1 3 3 -1 3 1 3 1 1 1 2 3 -2 -3 1

-2 3 -2 -3 -3 -3 -3 1 3 1 -2 -2 -3 -1 3 -1 -1 -2 -2 1 -2 -3 2 -3 -2 3 -2 -1 -2 -2 3 -2 - 3 -3 -1 -3 2 -1 3 -2 -1 -3 -2 -1 -2 -3 1 1 3 3

-1 3 -2 -3 -3 -2 -2 -2 3 2 -1 -2 3 1 3 -2 1 -2 3 1 1 3 -2 1 3 -1 -3 -3 1 -2 -2 3 -1 -2 -2 1 2 1 -2 -1 2 1 -2 1 -2 -3 -2 -3 1 2 -1 -2

-2 -3 -3 -2 -3 1 1 1 3 1 -3 -3 -1 -3 1 -3 -3 -2 -2 -1 -1 3 1 -2 -1 -1 2 3 -2 1 3 -1 -3 -2 -2 -2 3 -1 -3 -3 -3 -1 -1 -3 2 1 3 2 -3 -1

-1 -3 - 2 -1 3 -2 -3 1 2 2 -3 -3 2 -3 -3 1 -2 -2 -1 -1 3 -2 -1 3 -2 -2 -3 2 -3 -1 -1 -1 -2 -1 -2 -3 2 1 3 -1 -3 -1 3 1 -3 -1 -2 -1 -2

-3 2 1 2 1 1 2 2 3 1 2 -1 -3 1 1 3 2 1 2 3 2 -1 3 2 2 3 -1 -3 -2 -3 -1 2 -3 -3 2 -1 2 -1 -1 -2 -2 3 2 2 1 2 -3 -3 -3 -2 -2 -1 2 -3 -1

2 1 -2 1 -2 -3 -1 3 -2 1 3 1 1 1 1 -3 1 -3 -3 1 3 -1 -3 2 -1 3 2 2 1 1 -2 -2 -3 2 -3 1 -2 3 -1 -3 1 1 -2 -2 3 3 -1 2 1 -3 -2 3 -1 -2

3 3 -1 -2 -2 -1 -3 2 1 -3 2 -1 -3 2 -3 -2 1 2 -1 2 1 2 -1 -2 -3 -3 -1 -3 2 3 -2 -2 1 1 -3 2 -3 -3 1 -3 -3 2 -1 2 -3 -1 -2 1 1 -3 1 3 3

-1 2 3 1 3 3 2 3 3 -1 -3 -2 -3 - 3 -3 1 2 3 -1 -2 -3 -3 1 -3 2 -1 3 1 2 1 -3 -2 -1 -1 -3 -2 -2 -2 3 3 -1 2 2 3 1 -2 -3 -1 -2 1 3 1 2

-3 -3 -1 3 -2 1 -2 -2 -1 2 -3 -1 2 -3 -2 1 -2 -3 -1 -1 -1 -3 -1 -3 -2 -3 2 -3 -2 3 2 1 -3 -3 1 2 1 2 -3 1 -2 1 2 -3 1 3 1 -2 -1 -2 -2

-2 -3 -2 1 -2 3 1 1 3 2 -1 -3 -3 -2 -1 -1 2 -3 -1 3 -1 3 -1 -1 2 1 -2 1 3 1 -3 -2 1 1 -2 1 2 3 2 -3 1 2 -1 2 -3 -3 -3 2 -3 -1 -3 -2 -1

2 3 3 3 3 3 -1 -1 2 -1 -2 -3 -1 -2 1 1 2 1 1 1 1 2 3 3 1 -2 -2 1 2 1 3 2 -3 -1 -1 -2 1 1 3 2 -3 -2 3 1 -3 1 -2 -2 -2 3 3 -2 1 -2 1

-2 -2 -2 -3 -2 -2 -1 2 1 3 -2 -3 -1 -2 3 -1 -2 -1 2 -3 -1 3 -2 -3 2 1 -2 3 -1 -3 -1 2 3 -1 -2 1 2 -1 2 -3 -2 -3 1 3 -2 1 -3 2 3 3 -1

-1 -3 2 -1 -1 2 1 3 3 1 1 -2 -3 -3 -1 2 3 2 3 -1 -1 -2 3 -1 3 2 -1 -1 2 1 -2 3 2 -3 2 - 1 -3 -3 -2 1 3 -2 -2 3 -2 3 -2 1 2 2 -3 2 3 2

3 -1 3 2 -3 1 -3 1 3 -2 -2 -3 -1 - 3 1 3 1 -3 -2 1 -2 -3 2 3 3 -2 -1 2 1 -2 -2 -3 -2 -1 -2 3 -2 1 1 -2 -1 2 -1 2 1 -3 -2 3 1 -3 -2 -1

-1 -2 1 2 -1 -1 3 2 -3 -3 -3 -2 3 1 3 -2 -2 -1 -2 3 -1 -2 -2 3 3 -1 -3 -1 -3 2 -1 3 3 -2 3 -1 3 -1 -2 1 -3 -2 1 1 -2 -1 -3 1 3 -1 -2

-1 -1 -1 2 3 -1 3 -2 2 3 3 -1 -2 -2 -1 3 1 -3 2 -1 -1 -1 -1 3 3 -2 -3 -3 1 3 2 -1 -3 2 - 1 -3 -2 -3 -2 -1 -1 -2 3 -2 -3 -2 -2 -1 2 -3

-1 -1 2 -3 2 3 3 2 -3 2 3 -2 -3 -3 -3 -2 -3 -1 -2 -2 3 -2 1 -3 1 -3 -2 -2 3 2 -3 1 -2 -1 2 -3 2 -1 -3 -1 -3 2 -1 2 2 2 -1 2 1 3 1 3 1

1 1 2 -3 -1 -2 1 1 3 -1 -2 -2 -2 3 -2 -2 3 2 1 1 3 -1 -1 3 3 -1 -1 -2 3 2 2 -3 1 -2 -2 -1 -1 2 -3 -1 -3 -2 1 -3 2 2 1 2 -1 -3 1 2 -3

1 -3 -1 -1 -1 3 3 3 -1 -1 -1 2 2 1 2 2 2 3 -2 -3 -1 3 1 -3 -3 -2 -3 1 -2 -1 -1 2 3 -1 -3 -3 -3 -1 -2 -3 -2 3 3 -1 -1 -3 1 3 2 1 3 1 -3

2 3 -2 3 1 2 2 2 3 3 -1 -3 -1 -1 -3 -2 1 -2 3 -2 3 3 2 -1 3 2 1 -2 -2 1 1 1 2 3 3 2 1 3 1 1 -2 -3 -2 -1 2 -1 -1 3 2 -3 -1 -2 -3 -2 -1

-3 -3 -3 1 3 2 -3 -2 1 -2 -3 2 1 3 1 -2 1 1 2 1 3 1 -2 -2 1 2 -3 1 1 -2 -2 3 3 -2 1 2 3 -2 1 1 3 1 2 -1 -1 -1 -1 2 3 -1 -3 2 2 -3 2

2 2 3 - 1 -3 -1 -2 1 1 -2 1 3 3 -2 3 1 3 -1 -2 1 2 -3 1 1 1 1 3 -2 -2 1 -2 -1 2 -3 1 -3 2 -1 2 2 -1 -2 3 3 -2 3 -1 -1 2 1 3 -1 2 1 -3

-1 -1 3 1 3 -1 2 2 -3 -1 -2 -1 3 3 -2 1 3 -1 -2 -3 1 2 2 -3 -3 -2 -1 -3 1 3 2 2 -3 -2 -3 1 2 2 -3 -3 1 -2 1 -3 -3 2 -3 -3 1 1 2 1 -2

3 1 3 -1 3 -2 3 -2 3 3 3 -1 2 -1 -1 -3 1 2 1 3 -1 -3 -2 -3 1 3 1 -2 -3 -3 -2 1 2 -1 -3 -1 -2 -1 -2 1 -2 -3 -3 -1 3 3 -1 2 -3 1 -3 -3

-3 1 -3 2 2 1 -3 -2 -3 -1 -3 -3 1 3 -1 -1 -3 -2 -2 -3 -1 -1 3 1 2 2 -1 -2 -3 -1 2 3 -2 - 3 -1 3 -1 -3 2 1 -2 -2 1 -3 -2 1 1 3 2 -3 2 3

-1 3 -1 -2 -2 3 1 -3 -2 -2 -2 1 3 1 3 -1 2 2 3 3 1 2 1 -3 -1 3 -2 1 -3 -3 1 1 3 -1 3 1 -2 1 2 -1 2 3 3 1 2 2 3 -1 -3 2 3 2 2 1 3 2 2

-3 -3 2 1 1 3 3 3 1 1 -2 -3 2 3 2 -3 -1 -1 2 1 1 -3 2 3 -1 3 1 -2 3 1 -2 1 3 -2 -3 -3 -1 -1 3 -1 -3 -1 3 3 3 1 3 -2 3 -2 -1 -1 -1 3 3

-1 -1 2 1 -2 -3 1 -2 -2 -1 -2 -3 2 1 -2 -3 -2 3 -1 -1 -3 -1 -3 -2 1 3 2 3 -2 -2 -3 -3 - 1 2 2 2 3 3 2 1 1 -2 3 3 1 -3 -2 -3 -2 -1 2 1

3 1 -2 1 3 -1 -3 -2 3 -1 -1 2 2 -3 -3 1 3 1 3 3 1 -3 -3 -1 2 2 -1 -2 -3 1 2 2 -1 2 2 3 -1 -3 -1 2 -3 2 -1 -2 -2 -3 2 -3 -1 -1 -2 -1 -2

-3 2 1 1 2 3 -1 3 1 2 3 -1 2 -1 -3 -2 1 -2 -3 -1 -2 -3 1 2 -3 1 -2 3 3 -2 -1 -3 -2 1 3 -1 -1 -1 3 -1 3 2 -1 -2 -3 1 1 -3 -1 2 -1 -2 1

-2 -3 -3 -1 -2 -2 3 -2 3 2 2 2 2 -1 -2 -3 1 1 3 -1 2 1 -2 3 2 -1 2 2 3 2 -1 -2 -3 1 2 2 -3 -1 -3 1 -3 -3 -1 2 2 3 2 1 3 2 2 1 3 -2 -1

2 2 3 -1 2 -3 2 3 2 -3 -2 -1 2 -1 -2 -2 -3 -2 3 1 1 2 3 2 -1 -2 -3 1 1 3 1 3 2 3 3 -2 -2 -2 -2 3 -1 3 1 1 -2 -2 -1 -1 2 -1 3 1 -2 -1

2 2 2 2 2 -3 1 1 2 3 -1 -1 2 -1 3 1 2 -3 -2 -3 -2 1 -2 -3 1 3 1 -3 -1 3 1 2 -1 -1 2 -3 -1 3 2 -3 -2 -3 2 1 -3 2 1 1 1 3 3 -2 -1 2 3

3 -2 3 2 1 2 1 3 1 -3 -3 1 1 2 -1 -2 -1 -1 -1 -1 -3 -1 -3 -2 -1 -3 -1 -3 -1 -1 2 -1 2 3 1 -2 -2 3 3 1 2 -1 -3 -2 -1 -1 2 -1 3 -1 -3 -3

-3 -1 3 2 -3 -1 3 3 -2 -2 -3 -1 -2 1 -3 1 -2 -1 -2 -3 -1 3 -1 -2 1 3 1 1 1 -3 2 -1 -1 3 3 1 -3 -1 -3 2 2 -3 -3 -3 1 1 3 3 -2 -1 -3 -2

-3 -1 2 3 -1 -3 2 1 -2 -2 3 2 2 1 -3 -2 1 1 -2 3 1 -2 -3 -2 1 -3 1 -2 -3 1 3 2 1 -2 3 -2 1 -3 2 -1 -3 1 2 1 2 -3 -2 1 2 2 -1 -3 -3 2

-3 1 -3 1 -2 -1 -3 -1 -3 2 2 -1 2 3 -2 1 2 3 1 2 -3 1 -3 1 -3 -2 1 -3 -3 -2 1 2 -3 - 2 -1 3 -1 -3 2 -3 -1 -3 -2 1 2 -3 2 3 2 2 -3 -1 3

3 3 2 -3 1 3 2 1 -2 -2 3 1 3 1 2 2 2 1 2 2 -1 -2 1 -2 -2 -1 2 -1 3 -1 3 -1 -2 3 -1 2 -3 -1 2 2 3 -2 1 2 2 -1 2 2 2 -3 -3 -1 3 2 -1 3

-2 -2 -2 -1 -1 3 -1 -2 -1 -2 -2 1 -3 -3 2 3 3 -2 -1 -3 -3 2 -3 -3 1 2 -1 3 1 2 -3 -3 -3 2 -1 2 -1 -2 1 1 -3 -3 2 -1 2 1 -2 1 -2 -1 -2

```
1 3 -2 -2 -3 -1 2 3 -2 -3 2 3 -1 -2 -2 -2 3 2 2 -1 -3 2 3 -2 -3 1 -2 -1 -2 -2 -3 1 3 3 3 1 3 -2 -1 2 2 1 3 2 -1 2 -3 -2 -1 3 3 1 1 -2
-3 -1 -2 3 2 -3 -1 2 3 1 3 3 -1 -3 1 1 2 -1 2 2 2 1 3 -2 3 -1 -3 -3 1 3 -1 2 -3 -2 3 1 -3 -3 -2 -2 -3 -1 3 1 3 -1 -2 -1 2 1 1 1 -2 3
-1 -1 2 2 3 -1 -1 -3 1 -3 -2 1 1 -2 -1 3 3 1 1 3 -1 -2 1 2 -3 1 -3 -2 -1 -3 -1 -3 -2 -2 3 1 3 2 3 -2 -2 3 -1 -1 -2 1 1 3 -2 -3 -3 -1 2
3 -2 1 2 -1 -2 -1 -3 1 -2 1 -3 -3 -1 2 -1 2 1 3 2 3 -1 2 2 -1 -3 2 -3 1 3 2 -1 -2 -1 3 3 -1 2 1 -3 1 -2 3 3 3 2 -1 -2 -1 2 3 -1 3 1 3
-1 -2 3 2 -1 2 1 3 -1 -1 3 2 2 -1 2 -1 3 2 2 2 3 1 2 2 3 3 1 3 1 3 -1 2 2 -3 2 3 1 1 -2 3 2 -1 2 2 -3 1 1 3 1 3 -2 1 3 2 1 -3 -3 -2
1 3 1 3 1 2 3 -1 2 -1 2 -3 -3 -3 1 3 1 1 -3 -3 1 -3 -2 -1 -3 2 -3 -3 -2 1 -2 -3 1 2 -1 -3 1 1 2 1 -3 -1 -2 -3 2 1 2 3 2 -3 -1 -2 3 3
-2 3 -2 -1 -1 -2 1 3 -2 1 3 -2 1 3 3 2 3 -1 3 -2 -2 -3 -2 1 3 2 3 2 -1 -1 2 3 2 3 3 3 1 -2 1 -2 1 -2 -1 -2 -2 -2 1 2 2 1 2 2 -1 3 -1
-1 2 2 1 -3 -1 2 2 -3 -2 3 3 -2 -3 1 2 1 -2 -2 3 1 -3 -1 -3 -2 -1 -3 1 -3 1 2 1 1 -3 -3 1 3 1 3 2 2 1 2 -1 2 -1 2 -3 -3 1 1 1 1 3 1
2 -1 2 -1 3 -1 -3 2 -1 -2 3 -2 -2 1 3 2 -1 -2 -2 -1 -1 -2 -3 -1 2 -1 -2 1 1 -3 -2 -3 1 3 -2 3 2 -1 3 -1 -2 -3 -2 -1 -3 -2 -1 -2 3 3 -1
2 3 -2 -3 2 2 -3 -1 3 -2 1 -3 2 2 1 2 -3 -3 -2 1 1 -3 -2 -1 2 -1 -3 -1 3 1 -2 -2 3 -2 -3 -3 2 1 -2 -3 1 1 -3 1 1 3 -2 2 -2 1 3 2 -1
-2 -3 1 3 2 -1 -1 -1 -2 3 1 -2 -1 2 1 1 1 2 3 2 -1 -3 -3 2 -3 1 -3 -3 -1 -1 2 3 2 2 3 -2 -3 2 2 1 2 -3 2 -1 3 1 -2 -2 -1 -3 2 3 2 -1
-2 -1 -1 -2 -3 2 1 2 -3 2 -1 2 3 -2 3 -1 -2 -3 1 3 -1 -2 -1 -2 3 3 -2 -3 -2 1 2 2 2 2 -3 -2 3 -2 -2 -3 -3 -3 -2 -3 -3 1 -3 -1 -2 -1 3
1 2 -1 2 2 -1 -1 -1 -3 1 -2 3 2 2 -3 -2 -2 -3 -1 2 -1 -1 -3 -2 -2 3 2 -3 2 3 -2 3 - 1 3 -2 3 3 1 -2 -3 -3 1 1 3 -2 1 -3 -2 -2 -1 -2 1
-2 1 1 3 -2 -2 -1 3 -1 -3 -1 - 3 1 -2 -1 -3 1 3 2 1 2 -1 -2 -3 2 2 2 3 1 3 3 1 -3 -2 3 2 -1 -3 -2 -1 2 1 2 1 2 -1 -1 3 1 3 -2 3 -1 -2
-1 -1 -2 3 1 -3 -2 3 1 2 3 1 1 3 3 -1 -3 -2 -1 3 3 3 1 3 3 3 -1 3 -1 2 -3 -3 1 1 -2 -2 -1 -2 -2 3 -2 -2 3 1 -2 -3 -2 3 3 -1 -1 -3 1
-2 3 2 2 3 2 3 -2 -1 -3 2 2 3 2 1 3 -2 -1 -1 3 2 3 -1 -2 1 2 1 -3 -3 2 2 -3 -1 2 3 1 1 -3 1 -2 -1 -2 -1 2 3 2 2 1 -2 -1 -2 3 -2 -2 -3
-2 -2 1 -2 -3 -1 3 -2 -2 -3 -2 -3 -2 1 1 -3 -2 1 1 3 -2 3 3 -1 -2 3 1 2 2 1 -2 -2 1 -3 -2 -1 -3 -1 -2 -2 -1 3 1 1 -3 1 -3 -1 -2 3 1 -3
1 2 -3 -1 -1 -1 -2 -1 2 -3 -2 -2 -1 -3 -3 -2 3 2 3 3 3 - 1 -2 -2 -1 -3 1 -3 -1 -2 -3 1 -3 -3 2 -3 -3 2 -1 -1 -2 -3 2 3 -2 -3 1 1 -3 -1
-2 -2 -3 -3 2 1 3 2 2 2 1 3 -1 3 2 1 1 1 2 3 1 2 -1 -3 1 3 -1 -2 -2 -3 -2 -2 1 2 - 1 -3 -3 -1 3 -1 -3 1 -3 -1 3 1 3 -1 3 1 3 2 1 -2 1
-2 -3 -2 -3 1 -2 -2 3 1 1 -3 -3 1 -3 -1 -2 -2 1 -2 -2 -2 -3 -2 -3 -3 -1 -3 2 2 3 -1 -2 3 -2 -1 -2 -1 -3 -2 -1 -2 -3 -1 -3 -2 -3 -2 3 -2
-1 -2 -1 3 -2 1 1 -2 3 -2 -2 -3 2 3 -2 1 3 1 2 3 1 -3 2 3 -2 1 2 -3 -1 3 -2 -3 -2 -3 1 2 3 -2 -3 -1 -2 -2 1 3 -2 3 1 2 -1 -1 -2 1 3 -
1 -2 -3 -2 1 -2 -1 3 -2 -1 3 -1 -1 3 2 -1 -1 -1 -2 -3 -2 -2 1 2 3 1 -2 -3 -2 -2 -3 -1 -3 1 -2 3 2 2 -3 1 -3 -3 -3 -1 3 1 3 3 -1 -1 -2
1 2 1 -3 -3 -3 -1 -1 -2 -1 -1 2 1 2 2 2 2 3 -1 -3 1 3 -2 -2 -1 -3 -3 1 3 -2 3 1 -2 1 3 -1 -2 -2 -2 1 3 -1 -1 -3 2 1 2 -3 -2 -3 -2 1 3
-2 -3 1 3 2 1 -2 1 3 3 -1 -1 2 2 -3 2 3 1 2 1 2 1 -3 2 1 -3 1 -2 1 -3 -1 -1 3 -1 -1 2 1 -3 -3 2 1 3 -2 -2 1 3 1 1 2 2 -3 1 2 1 -3 -2
1 3 1 2 3 3 2 -3 1 1 -2 3 -1 -1 2 2 1 3 -1 -3 1 2 -3 -1 -1 -3 -1 -3 -2 3 3 1 3 -2 -1 -2 3 3 -1 2 2 2 3 2 -1 -2 -1 -3 2 3 3 -2 -1 3 2
-3 2 1 2 2 -1 3 -2 -1 -3 2 -1 -2 -1 -3 2 1 1 2 1 3 -2 -1 2 3 -1 2 1 -2 -1 3 3 1 -3 -3 -2 -2 3 2 3 2 3 2 1 -2 -1 -1 2 2 -3 1 -3 2 2 -1
3 -2 1 1 3 -1 3 3 3 1 3 -1 -1 -2 1 3 2 2 -1 -3 2 1 - 3 -3 -2 1 1 2 1 -3 2 2 -3 -2 3 -1 -3 -2 3 3 -1 3 3 -2 -2 3 3 -1 -3 2 2 -1 -3 -1
-1 -1 -3 -1 -3 -2 -1 -2 -1 -1 -2 3 1 -2 -1 -2 1 3 3 -1 3 2 -3 -3 -2 -2 3 1 1 2 2 1 -2 1 -2 -2 -2 -1 3 -1 -3 -3 -1 3 1 -3 1 3 1 3 -2 1
3 -2 -2 -3 2 1 -2 3 3 -1 - 1 -3 2 2 2 3 2 1 -2 -1 -1 3 3 1 2 3 -2 -2 -1 -3 1 3 1 -3 -3 -1 -1 -2 -1 2 3 3 -2 -2 3 1 1 3 -2 1 2 2 -1 -2
1 3 2 1 -3 -3 2 3 -1 3 2 -1 2 3 -2 -2 -2 -1 -2 -1 2 1 -2 3 -2 1 -3 -3 -3 2 -3 -3 -3 -1 -2 -1 -2 -1 2 -1 2 2 -1 -1 2 -3 -2 -3 -1 -3 2
-3 1 1 -3 -2 3 1 -3 -2 1 2 3 1 -2 -2 3 -1 2 3 1 2 -1 3 3 3 -2 -2 -1 -3 -1 3 2 1 -3 2 -3 2 -3 1 2 3 3 2 -3 2 1 -3 -2 -3 1 1 -3 -2 -2 1
1 2 -3 -1 -1 -2 -1 2 -1 -3 1 3 3 3 2 2 1 3 -1 -1 -2 -1 -2 -2 -3 -1 3 1 -2 1 -3 1 -3 1 3 3 1 3 2 3 -2 -3 2 -1 2 -1 3 -2 -2 -3 2 1 -3 2
-3 -1 -2 3 1 -3 -3 -1 2 1 -2 1 -2 1 2 2 -3 2 1 1 -2 3 -2 -1 3 2 3 -1 2 -3 -3 -2 -1 -3 -2 1 1 -3 2 1 2 1 -3 -3 2 1 -2 -1 -3 -2 1 1 2 1
2 2 -3 2 -1 -2 -3 -2 1 3 3 3 3 -1 -2 3 -2 -2 -1 3 3 1 -2 1 2 2 -1 -3 2 2 2 -1 2 -1 -2 3 1 1 3 -2 -3 -3 -3 -2 -1 -1 -1 -2 3 3 2 3 1 2
-1 2 3 2 2 3 3 2 1 1 3 1 2 -3 1 2 3 -2 1 -2 3 -2 3 3 3 3 3 -1 2 1 3 1 1 3 -2 -2 -1 -2 -2 1 1 -2 -3 1 2 2 -1 -3 -3 -1 -3 2 1 -2 1 -3
-2 1 -3 2 -1 -1 -1 2 1 -2 -2 1 -3 -1 -2 -2 -1 2 2 -1 3 -2 1 -3 1 2 2 -3 -2 3 -2 -2 1 1 2 1 1 -2 -1 2 -3 -1 -3 -2 3 -1 -1 -3 2 3 1 1 2
1 -3 -2 -1 -3 -3 -2 -1 -1 2 2 3 -1 3 3 -2 -3 1 1 -2 -3 -3 -3 -1 -1 2 -1 -2 -2 - 2 1 -3 -1 2 1 2 3 1 3 -1 2 -1 -3 1 1 1 3 1 3 1 -2 1 -2
-1 -2 -2 -1 -1 -1 -3 -1 2 -1 2 -1 2 1 -2 -3 -1 -1 -1 -1 3 -2 3 -1 -1 -2 -2 -1 3 -1 -2 -3 -2 -3 2 2 3 2 1 -2 1 3 -1 -2 -3 -2 -3 -3 -3 -2
-2 -2 -1 3 -2 -1 -3 -3 -1 -3 -1 -2 3 -2 -2 -3 -1 3 2 1 2 2 1 -3 -1 -3 -1 -3 -1 3 3 3 -2 -2 1 -2 -3 2 2 -1 3 -2 1 2 -3 -2 -1 -1 -3 -2 1
-3 2 1 3 -2 3 1 2 -1 3 -2 -2 1 2 3 2 -1 -2 -3 1 2 2 -1 3 3 -1 3 2 2 1 1 -3 1 3 3 3 -2 -2 -3 -2 3 -2 -1 2 2 2 2 3 3 -1 2 3 3 3 2 3 -1
-2 3 2 -1 3 -2 3 3 -2 -2 -1 -3 2 3 3 -2 -2 -1 -1 2 1 -2 -3 1 3 3 2 -1 -3 -1 -3 1 2 -1 -3 -2 3 -2 1 -3 1 -3 -2 1 -3 2 2 2 1 -3 -1 -1 2
```

-1 -3 -1 2 1 1 2 -1 3 -2 -3 1 2 2 1 -3 -1 -2 -3 -3 -1 3 1 2 3 1 -2 -1 -2 1 -3 1 -3 -3 -3 -2 1 -3 -3 -2 -1 3 1 3 1 -2 3 1 -2 -2 -2 1 1 3 -1 -3 2 -3 -2 1 -3 -2 1 1 -2 3 3 3 1 2 1 -2 1 -3 -2 -1 2 -3 -3 1 2 2 1 -3 -3 2 -3 -3 1 1 2 2 1 2 -3 2 -1 -1 -1 -2 3 -1 -2 -2 -2 3 -1 2 1 2 3 -2 3 3 2 - 1 2 3 2 3 -1 2 2 2 3 1 3 1 -2 3 1 1 3 3 -1 3 -1 -2 1 3 2 3 3 2 1 -2 -1 3 -1 -2 1 -2 1 -2 -2 3.

E.3 Collisions for Morgenstern hashes, $q = 2$ and $\deg p(x) = 20$

Now we give a small example for our collision-finding algorithm. The polynomial we choose to target is $p(x) = x^{20}+x^{17}+x^{14}+x^{13}+x^{12}+x^{11}+x^9+x^7+x^5+x^3+x^2+x+1$. We choose $R = 10$ and generate random m and b''. After 3 random trials we get $m = x^9+x^8+x^7+x^6+x^5+x^4$, $b'' = x^{10}+x^8+x^5+x^2+1$ so $k = 52$, $a = x^{52}+x^{48}+x^{36}+x^{32}+x^{30}+x^{25}+x^{24}+x^{22}+x^{20}+x^{15}+x^{14}+x^{12}+x^{11}+x^{10}+x^9+x^4+x^3+x+1$, $b = x^{51}+x^{50}+x^{48}+x^{47}+x^{46}+x^{45}+x^{44}+x^{40}+x^{39}+x^{38}+x^{37}+x^{36}+x^{35}+x^{34}+x^{32}+x^{31}+x^{30}+x^{29}+x^{28}+x^{27}+x^{25}+x^{24}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{11}+x^{10}+x^9+x^8+x^5+x^4+x^3+x^2+x$ and $n = x^{62}+x^{61}+x^{59}+x^{57}+x^{55}+x^{53}+x^{52}+x^{51}+x^{50}+x^{49}+x^{48}+x^{46}+x^{45}+x^{40}+x^{37}+x^{31}+x^{29}+x^{28}+x^{26}+x^{25}+x^{24}+x^{23}+x^{16}+x^{15}+x^{13}+x^{12}+x^{10}+x^6+x^5+x^3+1$.

The polynomial n has three factors $n_1 = x^{56}+x^{54}+x^{53}+x^{50}+x^{48}+x^{46}+x^{44}+x^{40}+x^{36}+x^{34}+x^{33}+x^{30}+x^{29}+x^{22}+x^{20}+x^{18}+x^{13}+x^{11}+x^7+x^6+x^5+x^3+1$, $n_2 = x^4+x^3+x^2+x+1$ and $n_3 = x^2+x+1$ which are all of even degrees. For each factor n_i we compute α such that $\alpha^2+\alpha+1 \equiv 0 \bmod n_i$ and use this value and the continued fraction algorithm to recover (c_i, d_i) such that $c_i^2+d_i^2+c_id_i \equiv 0 \bmod n_i$: we get $(c_1, d_1) = (x^{26}+x^{25}+x^{24}+x^{21}+x^{20}+x^{18}+x^{16}+x^{14}+x^{13}+x^{11}+x^8+x^6+x^5+x+1, x^{28}+x^{23}+x^{21}+x^{19}+x^{15}+x^{13}+x^{10}+x^7+x^5+x^4+x^2+x+1)$, $(c_2, d_2) = (x, x^2+1)$ and $(c_3, d_3) = (x, 1)$.

Combining these partial results we get $c = x^{51}+x^{50}+x^{47}+x^{41}+x^{40}+x^{36}+x^{31}+x^{27}+x^{26}+x^{25}+x^{24}+x^{23}+x^{22}+x^{20}+x^{18}+x^{17}+x^{16}+x^{14}+x^{12}+x^{11}+x^{10}+x^9+x^7+x^4$ and $d = x^{51}+x^{50}+x^{49}+x^{48}+x^{47}+x^{45}+x^{44}+x^{43}+x^{42}+x^{39}+x^{36}+x^{33}+x^{31}+x^{30}+x^{29}+x^{27}+x^{26}+x^{25}+x^{22}+x^{21}+x^{20}+x^{18}+x^{17}+x^{16}+x^{14}+x^{13}+x^9+x^7+1$.

We can verify that

$$(a^2+b^2+ab)+(c^2+d^2+cd)x = (1+x)^{2k}$$

and $(a, b, c, d) \equiv (1+x)^k(1, 0, 0, 0) \bmod p$. We factorize the lifted matrix and, using the indices of the generators given in Section C.2, we get the following collision with the void message: 0 2 0 2 0 1 2 1 2 1 2 1 2 0 2 0 2 0 2 0 2 0 1 0 2 0 2 1 0 2 1 0 1 0 2 1 0 1 2 1 2 1 2 1 0 2 1 0 1 2 0 1 0 1 0 1 0 2 1 0 1 2 0 2 1 2 0 2 0 1 2 0 2 0 1 0 2 1 2 1 0 2 0 1 0 1 2 0 2 0 2 0 2 0 1 2 1 0 2 0 2 1 0 1.

E.4 Collisions for Morgenstern hashes, $q = 2$ and $\deg p(x) = 1024$

Let $p(x) = x^{1024} + x^{1023} + x^{1022} + x^{1020} + x^{1014} + x^{1013} + x^{1009} + x^{1006} + x^{1003} +$
$x^{999}+x^{993}+x^{992}+x^{990}+x^{989}+x^{988}+x^{987}+x^{986}+x^{983}+x^{982}+x^{981}+x^{980}+x^{979}+$
$x^{977}+x^{976}+x^{971}+x^{967}+x^{965}+x^{961}+x^{960}+x^{957}+x^{955}+x^{953}+x^{946}+x^{945}+x^{943}+$
$x^{941}+x^{937}+x^{936}+x^{935}+x^{934}+x^{930}+x^{925}+x^{923}+x^{920}+x^{919}+x^{918}+x^{917}+x^{915}+$
$x^{914}+x^{911}+x^{910}+x^{909}+x^{908}+x^{906}+x^{904}+x^{901}+x^{900}+x^{899}+x^{898}+x^{896}+x^{895}+$
$x^{894}+x^{889}+x^{888}+x^{885}+x^{884}+x^{882}+x^{878}+x^{876}+x^{875}+x^{872}+x^{870}+x^{866}+x^{864}+$
$x^{863}+x^{859}+x^{857}+x^{856}+x^{855}+x^{854}+x^{851}+x^{850}+x^{849}+x^{846}+x^{838}+x^{837}+x^{834}+$
$x^{831}+x^{830}+x^{829}+x^{828}+x^{827}+x^{821}+x^{818}+x^{813}+x^{812}+x^{810}+x^{809}+x^{808}+x^{807}+$
$x^{806}+x^{805}+x^{804}+x^{803}+x^{802}+x^{800}+x^{799}+x^{798}+x^{796}+x^{795}+x^{793}+x^{791}+x^{788}+$
$x^{785}+x^{784}+x^{783}+x^{781}+x^{776}+x^{775}+x^{773}+x^{771}+x^{770}+x^{769}+x^{768}+x^{766}+x^{760}+$
$x^{753}+x^{751}+x^{749}+x^{747}+x^{745}+x^{743}+x^{742}+x^{735}+x^{734}+x^{733}+x^{732}+x^{730}+x^{729}+$
$x^{726}+x^{724}+x^{722}+x^{719}+x^{718}+x^{716}+x^{715}+x^{712}+x^{711}+x^{707}+x^{706}+x^{705}+x^{700}+$
$x^{696}+x^{695}+x^{693}+x^{692}+x^{690}+x^{685}+x^{681}+x^{676}+x^{675}+x^{674}+x^{673}+x^{671}+x^{670}+$
$x^{669}+x^{664}+x^{662}+x^{661}+x^{658}+x^{656}+x^{654}+x^{652}+x^{651}+x^{650}+x^{649}+x^{648}+x^{646}+$
$x^{645}+x^{643}+x^{641}+x^{640}+x^{639}+x^{637}+x^{635}+x^{634}+x^{633}+x^{632}+x^{631}+x^{629}+x^{628}+$
$x^{626}+x^{624}+x^{623}+x^{621}+x^{619}+x^{615}+x^{612}+x^{611}+x^{605}+x^{604}+x^{603}+x^{600}+x^{598}+$
$x^{596}+x^{594}+x^{590}+x^{588}+x^{586}+x^{585}+x^{582}+x^{579}+x^{577}+x^{571}+x^{570}+x^{564}+x^{562}+$
$x^{561}+x^{559}+x^{558}+x^{557}+x^{556}+x^{550}+x^{549}+x^{545}+x^{544}+x^{541}+x^{540}+x^{538}+x^{537}+$
$x^{535}+x^{534}+x^{528}+x^{526}+x^{525}+x^{524}+x^{520}+x^{519}+x^{518}+x^{516}+x^{515}+x^{513}+x^{512}+$
$x^{510}+x^{509}+x^{507}+x^{503}+x^{498}+x^{496}+x^{495}+x^{492}+x^{491}+x^{490}+x^{489}+x^{484}+x^{483}+$
$x^{481}+x^{480}+x^{478}+x^{477}+x^{476}+x^{475}+x^{474}+x^{473}+x^{468}+x^{467}+x^{465}+x^{464}+x^{463}+$
$x^{459}+x^{457}+x^{456}+x^{455}+x^{454}+x^{449}+x^{447}+x^{444}+x^{443}+x^{442}+x^{438}+x^{435}+x^{434}+$
$x^{431}+x^{429}+x^{427}+x^{425}+x^{424}+x^{415}+x^{412}+x^{411}+x^{409}+x^{406}+x^{404}+x^{403}+x^{402}+$
$x^{399}+x^{398}+x^{394}+x^{393}+x^{392}+x^{390}+x^{389}+x^{387}+x^{386}+x^{385}+x^{384}+x^{382}+x^{381}+$
$x^{380}+x^{379}+x^{377}+x^{374}+x^{373}+x^{369}+x^{368}+x^{365}+x^{362}+x^{357}+x^{354}+x^{351}+x^{349}+$
$x^{346}+x^{345}+x^{344}+x^{343}+x^{340}+x^{337}+x^{331}+x^{330}+x^{328}+x^{326}+x^{324}+x^{323}+x^{322}+$
$x^{321}+x^{319}+x^{317}+x^{315}+x^{314}+x^{313}+x^{312}+x^{310}+x^{309}+x^{305}+x^{304}+x^{303}+x^{298}+$
$x^{296}+x^{294}+x^{290}+x^{289}+x^{288}+x^{283}+x^{282}+x^{281}+x^{279}+x^{276}+x^{275}+x^{273}+x^{271}+$
$x^{268}+x^{266}+x^{265}+x^{264}+x^{263}+x^{260}+x^{259}+x^{253}+x^{252}+x^{250}+x^{249}+x^{247}+x^{246}+$
$x^{245}+x^{244}+x^{242}+x^{237}+x^{235}+x^{234}+x^{231}+x^{228}+x^{227}+x^{225}+x^{222}+x^{218}+x^{217}+$
$x^{216}+x^{215}+x^{214}+x^{211}+x^{210}+x^{208}+x^{206}+x^{204}+x^{203}+x^{202}+x^{201}+x^{200}+x^{199}+$
$x^{198}+x^{195}+x^{194}+x^{192}+x^{191}+x^{189}+x^{187}+x^{185}+x^{184}+x^{181}+x^{180}+x^{179}+x^{172}+$
$x^{171}+x^{170}+x^{165}+x^{164}+x^{162}+x^{161}+x^{159}+x^{157}+x^{153}+x^{152}+x^{151}+x^{149}+x^{148}+$
$x^{146}+x^{145}+x^{143}+x^{140}+x^{137}+x^{134}+x^{133}+x^{128}+x^{127}+x^{125}+x^{123}+x^{121}+x^{120}+$
$x^{119}+x^{115}+x^{113}+x^{110}+x^{108}+x^{107}+x^{105}+x^{103}+x^{102}+x^{100}+x^{99}+x^{96}+x^{94}+$
$x^{89}+x^{87}+x^{86}+x^{83}+x^{82}+x^{81}+x^{79}+x^{77}+x^{76}+x^{73}+x^{70}+x^{69}+x^{68}+x^{64}+x^{62}+$
$x^{61}+x^{59}+x^{57}+x^{56}+x^{55}+x^{54}+x^{53}+x^{51}+x^{50}+x^{49}+x^{48}+x^{47}+x^{45}+x^{44}+x^{41}+$

$x^{40}+x^{39}+x^{37}+x^{34}+x^{31}+x^{27}+x^{22}+x^{18}+x^{14}+x^{10}+x^{9}+x^{8}+x^{5}+x^{2}+x+1.$

Then the following sequence collides with the void sequence:

```
    0 2 0 1 0 2 0 2 0 1 2 1 0 1 0 2 1 0 2 0 1 2 0 1 0 2 1 0 2 0 1 0 2 1 0 1 2 0 2 0 2 0 2 1 0 2 0 1 2 1 2 0 1 2 0 2 0 1 2 1 0 2
1 0 1 2 1 2 0 2 0 2 1 2 1 2 0 1 2 0 2 0 1 2 0 1 2 1 2 1 2 1 0 2 1 0 1 2 0 1 0 1 0 2 0 2 1 2 0 2 0 1 2 0 1 0 2 1 2 0 2 1 0 2 0 2 0
1 2 0 1 0 2 1 0 2 0 1 2 0 2 1 2 1 2 1 2 0 2 0 1 2 1 2 0 2 1 0 2 1 0 1 0 1 2 0 1 0 1 0 2 0 2 0 2 0 1 0 2 0 2 1 0 2 1 2 0 1 0 2 0 1
2 0 1 2 0 2 1 2 1 0 2 1 2 0 1 2 1 0 2 1 0 1 0 2 0 1 2 0 2 0 1 2 0 2 0 1 2 0 2 0 1 2 1 2 1 2 1 0 2 0 2 0 2 1 2 1 0 2 0 1 0 2 0 1 0
2 1 2 1 0 2 0 1 2 0 1 0 2 1 0 2 0 2 1 2 0 1 2 1 2 1 0 1 0 2 1 0 1 2 0 2 0 1 2 0 1 2 1 0 2 0 2 0 2 1 0 2 0 1 2 0 1 0 1 2 0 2 0 1 0
1 2 1 0 1 0 2 1 0 1 2 0 1 2 0 2 0 2 0 1 2 0 2 1 0 1 0 1 0 1 2 0 2 0 2 1 2 1 0 2 0 2 0 1 2 1 0 1 0 1 2 1 2 1 0 1 0 1 2 1 0 1 0 2 1
0 2 0 1 0 1 0 2 0 2 0 1 2 0 2 1 2 1 0 2 1 2 0 1 2 1 0 2 0 2 0 2 1 0 1 2 1 0 1 0 2 0 2 1 0 2 1 2 0 1 0 2 0 2 0 2 1 2 1 2 1 0 2 1 2
1 0 2 1 2 0 2 0 1 0 2 1 2 0 1 2 1 2 0 2 0 1 0 2 0 2 1 2 0 2 1 0 2 0 2 1 0 1 2 0 2 0 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 0 2 1 0 2 0 1 2
1 2 1 2 1 0 2 1 2 0 1 0 2 0 2 0 1 0 1 0 2 1 0 1 0 1 0 2 0 2 0 2 1 2 1 0 2 0 1 2 0 2 0 1 2 1 2 0 1 0 2 0 1 0 2 0 1 2 1 0 2 0 1 0 1
2 0 2 1 0 1 0 1 0 1 2 1 0 2 1 2 1 0 1 0 2 0 1 2 1 0 1 0 1 0 2 0 2 1 2 1 0 1 0 1 0 1 0 2 1 0 2 0 1 0 2 0 1 0 2 0 1 2 1 2 1 0 1 2 0
2 1 2 1 2 1 0 1 0 1 0 1 0 1 2 1 0 2 0 1 2 1 2 0 1 0 2 0 2 0 1 0 2 0 2 1 2 1 0 2 1 2 0 1 2 1 2 0 2 1 2 0 1 2 1 0 1 0 1 2 1 2 0 1 2
1 2 0 1 0 2 0 1 0 2 1 2 0 2 0 2 1 2 1 0 1 2 0 2 0 1 0 1 0 2 0 1 2 0 1 2 0 1 0 1 2 1 2 0 2 1 2 0 2 0 1 0 2 1 2 1 0 2 1 0 1 2 0 1 0
1 2 0 2 0 2 0 1 0 1 2 0 2 1 0 1 0 1 2 0 1 2 1 0 2 0 2 0 2 0 1 2 0 1 0 1 0 1 0 1 0 2 0 1 0 2 1 0 1 2 0 1 2 1 0 1 0 1 0 2 0 1 2 0 2
0 1 2 1 0 1 2 0 1 2 0 1 0 2 1 2 0 1 0 1 0 1 2 0 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 0 2 1 0 1 0 1 2 1 2 0 2 1 2 0 2 0 2 0 1 0 2 0 2 0 1
2 1 0 2 1 2 0 1 0 1 2 1 0 1 2 1 0 1 0 2 1 2 0 1 0 1 0 2 1 0 1 0 2 1 0 1 0 2 1 2 0 1 0 1 2 1 0 2 0 2 0 1 2 1 0 2 1 2 0 1 2 0 1 2 1
2 1 0 1 0 2 1 2 0 1 0 2 1 2 0 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 0 2 1 0 2 1 2 1 0 1 2 1 2 1 2 1 0 2 1 2 0 2 0 2 0 2 0 1 0 2 0 1 2 0 1
2 0 1 0 1 0 1 2 1 0 1 0 2 0 2 1 0 2 1 0 1 2 1 0 1 2 0 2 1 2 0 2 0 1 2 0 2 1 2 0 2 0 2 0 1 0 1 2 0 2 1 0 2 0 1 2 1 2 1 2 0 1 0 1 2
0 2 1 0 2 1 0 2 1 0 1 2 0 2 1 2 1 2 1 0 2 1 0 1 2 0 1 2 0 1 2 0 1 2 1 2 1 0 2 0 1 2 0 2 0 2 0 2 1 2 1 2 1 2 0 1 0 2 0 2 1 2 0 2 0
1 2 0 1 0 2 0 2 0 1 2 0 2 1 0 1 0 2 1 2 1 2 1 2 0 1 0 2 0 1 0 1 0 1 2 0 2 0 2 0 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 0 2 0 1 2 1 2 1 0
2 0 1 2 1 0 2 1 2 0 2 0 1 2 1 2 0 1 0 2 1 2 0 2 0 2 0 1 2 0 2 0 2 1 0 1 2 0 2 1 0 2 0 1 2 0 1 2 1 0 1 0 1 2 1 0 2 0 2 1 2 1 2 0 2
1 2 0 2 0 1 2 0 1 0 2 1 2 1 0 1 0 2 0 2 0 2 1 2 1 0 2 1 2 1 0 2 0 2 1 0 2 1 2 0 2 1 2 0 1 2 0 2 0 2 1 0 1 0 1 0 1 0 1 0 1 0 2 1 0
1 0 2 0 2 0 2 0 1 2 0 1 2 1 2 0 2 1 2 0 1 2 1 0 1 2 1 2 1 0 1 2 0 1 0 2 1 2 1 0 1 0 2 0 2 1 0 1 0 2 0 1 2 1 2 1 2 1 0 2 1 0 1 0 1
0 1 0 1 2 0 2 1 2 1 2 0 1 2 0 1 2 1 0 2 1 2 0 1 0 2 0 2 0 1 2 0 1 0 1 2 1 2 0 1 0 1 2 0 2 0 2 1 2 0 2 0 1 2 0 1 0 1 2 1 2 0 1 0 2
0 2 0 1 2 1 0 2 1 2 0 2 1 0 1 2 0 2 1 2 0 1 2 0 1 0 2 0 1 2 1 0 2 1 0 1 0 1 0 2 1 0 1 2 1 0 2 0 2 1 0 2 0 2 1 2 1 0 1 0 1 2 0 2 0
2 1 0 2 0 2 1 2 0 1 2 1 0 2 0 2 1 0 1 0 1 2 1 0 2 0 2 1 0 2 1 2 1 2 1 0 2 1 2 0 2 0 2 0 2 0 2 1 2 1 2 0 2 0 2 1 0 1 0 1 0 1 0 2 0
1 0 2 1 0 2 0 2 1 2 0 2 1 2 0 1 0 2 0 2 0 1 2 1 2 0 1 2 1 2 1 2 0 2 1 2 0 1 2 0 1 2 1 0 1 0 1 0 2 1 0 1 2 1 2 1 0 1 0 1 2 0 2 0 2
0 1 0 2 0 1 2 0 1 2 0 1 0 1 2 0 1 0 1 2 1 0 2 0 2 1 0 2 0 1 0 1 2 1 0 2 1 2 1 2 1 0 1 0 1 0 2 1 2 0 1 2 1 2 0 1 0 1 2 1 0 2 0 1 2
0 2 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 1 2 0 2 0 1 2 1 0 1 0 1 0 1 2 0 2 1 2 1 2 0 1 2 1 0 2 1 0 1 0 1 2 1 0 2 1 2 1 2 0 2 1 2 1 0 1 2
1 0 2 0 1 2 1 2 0 2 0 1 0 1 2 1 2 0 1 2 0 2 0 1 2 1 0 1 2 0 1 0 1 0 2 1 2 1 2 0 2 1 0 2 0 2 1 2 1 0 1 2 1 2 0 1 0 2 1 2 1 2 1 0 1
2 1 0 1 2 1 0 1 0 2 1 0 2 1 2 0 2 0 1 0 2 0 1 0 1 0 2 1 2 0 1 2 0 2 0 2 0 2 1 0 2 0 1 2 1 0 2 0 2 0 2 1 0 2 1 0 2 1 0 2 1 2 1 2 0
2 1 0 2 0 1 0 1 0 2 0 1 2 1 0 2 0 1 2 0 2 1 0 2 1 0 1 2 1 2 0 1 0 2 0 2 0 1 0 2 1 2 0 2 0 1 0 1 0 2 0 2 0 1 2 1 0 1 0 1 2 1 2 1 0
2 1 2 0 1 0 2 0 2 1 0 2 1 2 1 0 2 0 2 0 1 2 1 2 0 2 1 2 0 2 1 2 1 2 0 1 0 2 0 2 1 2 0 1 0 1 2 0 1 2 0 1 0 1 2 1 0 1 2 1 2 1 2 0 1
0 2 0 2 0 1 0 1 2 0 2 1 2 1 0 1 2 1 0 1 0 1 0 2 1 0 2 0 2 0 2 1 2 1 0 2 1 0 1 0 2 0 1 2 0 2 0 2 0 1 0 1 2 0 2 0 2 0 1 0 2 0 1 0 2
0 2 1 2 1 2 0 1 2 1 0 2 0 2 0 2 1 2 1 0 2 1 0 1 2 1 0 2 1 2 1 2 0 2 1 2 0 1 2 1 0 1 0 1 2 0 1 0 2 0 1 0 2 0 1 0 1 0 2 0 1 0 2 1 2
0 1 0 2 0 2 0 1 2 1 2 1 2 0 1 0 2 1 2 1 0 1 2 0 2 0 1 0 1 0 1 2 1 2 1 0 2 1 2 1 2 1 2 0 2 0 1 2 0 1 2 1 0 2 1 0 2 1 0 1 2 1 2 1 0
2 0 2 1 2 1 0 1 0 1 2 1 0 2 1 2 1 2 0 2 1 2 0 1 0 2 0 1 0 1 0 1 2 1 0 2 0 1 2 0 1 0 1 0 1 0 1 0 1 2 1 2 0 2 1 2 1 2 1 2 0 2 1 2 0
1 2 0 2 1 0 1 0 1 0 1 2 0 1 2 0 2 1 0 1 2 1 2 0 1 0 2 1 0 1 2 0 2 1 0 2 0 1 2 0 2 0 2 1 2 0 2 0 1 0 2 0 1 0 2 0 1 2 1 0 2 0 2 0 2
1 2 0 1 0 2 0 1 0 2 0 1 0 1 2 1 2 0 2 1 2 1 0 2 0 1 0 1 0 1 0 1 2 0 2 1 0 1 2 1 2 1 2 0 2 1 0 1 0 2 0 2 1 2 0 1 2 0 1 2 0 1 2 1 0
2 1 0 1 2 0 1 0 2 0 1 2 0 2 0 2 1 0 1 0 1 0 1 0 2 1 2 1 0 2 1 0 2 0 2 1 0 1 2 0 2 1 2 0 1 0 2 0 2 1 2 1 0 2 1 2 1 0 1 0 1 2 1 2 0
```

```
2 0 1 0 1 0 2 0 2 1 0 2 0 1 0 1 2 1 2 0 2 1 2 1 2 0 1 0 1 0 2 0 2 0 2 0 1 0 2 0 1 2 1 2 0 1 2 0 1 2 1 0 2 0 2 0 1 0 1 0 2 0 1 0 1
0 1 2 1 0 2 1 2 0 2 0 2 0 2 1 2 0 2 1 2 1 2 1 0 2 1 2 0 1 2 1 0 2 1 2 0 1 0 2 1 2 1 0 1 0 2 1 2 1 0 1 2 1 2 0 2 1 0 1 2 0 2 1 2 0
1 0 2 1 2 0 2 1 0 2 0 2 0 2 0 1 2 1 0 1 2 0 1 2 1 2 1 0 1 2 0 1 0 1 2 0 2 0 2 1 0 1 2 1 2 1 2 0 2 1 0 1 2 0 1 0 1 0 2 1 0 1 0 1 2
1 2 1 0 1 2 1 0 1 0 2 1 2 0 2 0 1 2 1 2 0 2 0 1 2 0 1 2 0 2 0 2 0 2 1 2 1 0 1 2 1 2 0 1 0 2 1 0 1 0 1 2 1 0 2 0 1 2 0 2 0 2 0 2 0
2 1 0 2 0 1 2 0 2 1 0 2 1 0 1 0 2 1 2 0 2 0 1 0 1 2 1 0 1 0 1 2 1 0 1 2 1 2 0 2 0 1 0 1 2 1 0 1 0 1 2 0 2 1 2 0 2 0 2 1 2 0 2 1 0
2 1 0 2 0 1 2 0 2 1 2 0 2 0 1 2 1 0 2 1 2 1 0 2 1 0 2 0 1 2 0 1 0 1 0 1 0 1 2 0 1 0 2 0 2 1 2 1 0 1 2 0 2 0 2 1 0 1 0 2 1 2 0 1 2
1 0 1 0 2 1 0 2 1 2 0 2 1 2 1 2 0 2 1 0 2 0 1 2 1 0 1 0 1 0 2 1 2 1 2 1 0 1 2 0 2 0 2 0 2 1 0 1 0 2 1 0 2 0 1 2 1 2 0 2 1 0 2 0 1
2 1 0 1 0 1 2 0 2 1 2 0 2 0 2 1 0 2 1 0 1 2 1 2 1 2 0 2 0 2 1 0 1 2 0 2 1 0 2 0 2 1 2 0 1 2 0 2 0 2 0 2 1 2 0 1 2 1 0 1 0 2 0 2 1
2 0 1 2 1 2 0 1 0 2 1 0 1 2 0 1 0 1 0 1 0 2 0 2 0 1 2 1 0 1 2 0 1 0 1 2 0 2 0 1 2 1 0 1 2 0 1 2 1 2 1 2 0 2 0 2 0 2 1 0 2 0 1 2 0
2 0 2 1 0 1 0 2 0 1 2 1 2 1 0 2 0 2 0 2 1 0 2 1 2 0 2 0 2 1 2 1 2 1 0 2 0 2 1 2 0 1 2 0 2 0 1 2 1 2 0 2 0 2 0 1 2 0 2 1 2 1 0 1 0
2 0 1 0 1 0 2 1 0 1 2 1 2 1 0 1 0 1 0 1 0 1 2 0 1 2 0 2 1 0 1 2 0 2 1 2 1 0 2 0 1 2 1 0 2 1 0 2 0 1 0 1 0 2 0 2 0 1 0 1 0 1 0 1 2
1 2 0 1 2 0 1 0 1 2 0 2 0 2 0 1 2 1 2 0 1 2 1 0 1 0 1 0 2 1 0 2 1 0 2 0 2 0 2 1 0 1 0 1 2 1 2 0 2 0 1 0 1 0 1 2 0 1 0 1 0 2 1 2 0
2 1 2 0 2 0 2 1 0 1 0 1 0 1 2 0 1 2 0 2 0 1 2 0 2 0 1 2 1 2 0 2 0 2 1 0 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 0 1 2 0 2 0 2 0 2 1 0 2 1 2
1 0 2 0 1 2 0 2 0 1 0 2 1 0 1 0 1 2 0 1 0 2 0 1 2 0 1 2 0 1 2 1 2 1 0 1 0 2 1 0 1 2 0 2 0 1 0 1 0 1 0 1 0 2 1 0 2 1 0 1 0 2 0 2 0
1 2 0 2 0 2 0 2 1 2 0 2 0 1 0 2 0 1 2 1 2 1 2 0 2 1 2 1 2 0 1 2 0 1 0 2 0 2 1 2 1 0 2 0 2 0 2 0 2 0 1 2 1 0 2 1 0 1 2 1 0 1 0 1 0
2 0 1 0 1 2 0 2 1 2 1 2 0 2 1 0 2 0 1 0 1 0 1 2 0 1 2 1 2 0 1 2 0 2 1 2 1 0 1 0 2 0 2 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 0 1 0 1 0 2 1
0 1 2 0 1 2 1 2 1 2 0 1 0 1 0 2 1 2 0 1 0 1 0 2 0 1 0 2 0 2 1 2 1 2 0 2 0 2 0 2 1 0 1 2 1 2 0 1 0 1 2 1 0 2 0 2 0 1 0 1 0 2 0 2 1
2 1 0 1 0 2 1 0 2 1 2 1 2 0 2 0 2 0 2 1 2 0 1 0 2 1 2 0 2 1 2 0 1 0 2 1 2 0 1 2 1 0 2 0 2 1 2 0 2 1 0 1 2 1 0 2 0 1 0 1 0 2 1 2 0
1 0 1 0 1 2 0 2 1 2 0 1 2 0 1 2 0 1 0 2 1 2 1 0 1 2 1 2 1 2 0 2 0 1 2 0 1 2 0 2 0 2 1 0 1 0 2 0 1 0 2 0 1 2 1 0 1 2 1 0 2 1 2 1 0
2 0 1 0 1 2 0 1 0 1 2 1 0 2 1 2 1 0 2 0 1 2 1 0 2 0 1 0 2 0 2 0 1 0 2 0 1 2 0 1 2 1 2 0 1 0 1 2 1 2 0 1 0 2 1 2 1 2 1 2 1 2 0 2 1
0 1 2 1 2 0 1 0 2 0 1 0 1 2 1 2 0 1 2 1 0 2 1 0 1 2 0 1 2 0 1 0 2 1 2 1 2 1 0 1 0 2 0 2 0 2 1 2 0 1 2 1 0 2 1 2 1 2 0 1 0 2 1 0 2
1 2 0 1 2 1 2 1 0 1 0 2 1 0 2 0 1 2 1 2 0 2 1 2 1 2 0 2 0 2 1 0 2 1 0 2 0 2 0 1 2 1 0 2 0 1 2 0 1 0 1 0 1 0 1 0 1 0 1 2 0 1 2 1 0
2 0 1 2 1 2 1 0 2 0 1 0 2 1 0 2 1 0 1 0 1 0 2 0 1 2 1 0 1 0 1 0 1 0 1 0 2 1 2 1 2 1 0 2 1 0 1 0 2 1 2 1 2 0 2 0 1 0 2 0 1 2 0 2 1
2 1 2 0 1 2 0 2 1 0 1 2 1 0 2 0 2 1 2 0 1 2 1 0 2 1 0 1 0 2 1 2 1 2 0 2 1 0 2 0 2 1 2 1 0 1 0 2 0 1 2 1 2 1 2 1 0 2 1 2 0 2 1 0 1
2 0 1 2 1 0 2 0 1 2 1 2 0 1 2 0 2 0 1 0 2 0 2 1 0 1 0 1.
```

Index