Cryptographic Hash Functions in Groups and Provable Properties

THÈSE N^{O} 5250 (2011)

PRÉSENTÉE LE 16 DÉCEMBRE 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE CRYPTOLOGIE ALGORITHMIQUE
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Juraj Šarinay

acceptée sur proposition du jury:

Prof. J.-P. Hubaux, président du jury
Prof. A. Lenstra, directeur de thèse
Prof. A. May, rapporteur
Dr M. Stam, rapporteur
Prof. S. Vaudenay, rapporteur

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

Svetlane

Résumé

Nous considérons plusieurs fonctions de hachage “prouvablement sûres” calculant de simples sommes dans un groupe bien choisi $(G, *)$. Les propriétés de sécurité de telles fonctions se traduisent prouvablement et de manière naturelle en des problèmes calculatoires sur G, qui sont simples à définir et également potentiellement difficiles à résoudre. Étant donnés k listes disjointes L_i d’éléments du groupe, le problème de la k-somme consiste à trouver un $g_i \in L_i$ tel que $g_1 * g_2 * \ldots * g_k = 1_G$. La difficulté de ce problème dans divers groupes respectifs découle de certaines suppositions “standard” en cryptologie à clef publique, telles que la difficulté de la factorisation des entiers, du logarithme discret, de la réduction de réseaux, ou du décodage par syndrome. Nous exposons des indices montrant que le problème de la k-somme puisse être encore plus difficile que ceux susmentionnés.

Deux fonctions de hachage basées sur le problème de la k-somme, FSB et SWIFFTX, ont été soumises au NIST comme candidates pour le futur standard SHA-3. Les deux candidatures étaient appuyées par une quelconque preuve de sécurité. Nous montrons que l’estimation des niveaux de sécurité dans ces candidatures est sans rapport avec les preuves fournies. Les revendications en matière de sécurité ne sont soutenues que par des considérations sur des attaques existantes. En introduisant des bornes de “second ordre“ sur les bornes de sécurité, nous montrons les limites d’une telle approche de la sécurité prouvable.

Un problème dans la manière de quantifier la sécurité n’implique pas nécessairement un problème avec la sécurité elle-même. Bien que FSB traîne une histoire d’échecs, des versions récentes des deux fonctions susmentionnées se sont montrées bien résistantes aux efforts de cryptanalyse. Ceci, ainsi que les multiples connexions avec d’autres problèmes standard, indiquent que le problème de la k-somme dans certains groupes pourrait être considéré comme difficile *en lui-même*, et peut-être conduire à des bornes de sécurité prouvables. La complexité de l’algorithme de l’arbre non-trivial est en train de devenir un outil standard pour en mesurer la difficulté associée.

Nous proposons des modifications du Very Smooth Hash multiplicatif, et dérivons la sécurité des k-sommes multiplicatives, contrairement aux réductions originales apparentées à la factorisation ou au logarithme discret. Bien que les réductions originales demeurent valides, nous mesurons la sécurité d'une manière nouvelle et plus agressive. Ceci nous permet d'assouplir les paramètres et de hacher plus vite. Nous obtenons une fonction qui n'est que trois fois plus lente que SHA-256, et que nous estimons offrir au moins une résistance équivalente aux collisions. La vitesse peut être doublée par l'usage d'un module spécial, la sécurité de la fonction modifiée est justifiée uniquement par la difficulté des k-sommes multiplicatives modulo une puissance de deux.

Nos efforts culminent en une nouvelle fonction de k-sommes multiplicatives, qui généralise encore plus loin le design de Very Smooth Hash. Contrairement aux variantes antérieures, la consommation mémoire de la nouvelle fonction est négligeable. L'instance la plus rapide de la fonction est supposée offrir une résistance à la collision sur 128 bits en 24 cycles par byte sur un processeur Intel Core i7, et s'approche de la valeur de 17.4 offerte par SHA-256.

Les nouvelles fonctions proposées dans cette thèse n'atteignent pas de propriétés de sécurité habituelles telles la résistance à la pré-image ou à la collision, prouvables à partir d'une supposition largement établie. Elles bénéficient cependant d'une séparation inconditionnelle et prouvable des entrées en collision. Des changements dans l'entrée, de petite taille par rapport à une mesure bien définie, sont garantis de ne jamais produire le même résultat à travers la fonction de compression.

Mots-clés : cryptographie, fonctions de hachage, sécurité prouvable, problème des anniversaires généralisé, cryptosystèmes du sac à dos

Abstract

We consider several “provably secure” hash functions that compute simple sums in a well chosen group $(G, *)$. Security properties of such functions provably translate in a natural way to computational problems in G that are simple to define and possibly also hard to solve. Given k disjoint lists L_i of group elements, the k-sum problem asks for $g_i \in L_i$ such that $g_1 * g_2 * \ldots * g_k = 1_G$. Hardness of the problem in the respective groups follows from some “standard” assumptions used in public-key cryptology such as hardness of integer factoring, discrete logarithms, lattice reduction and syndrome decoding. We point out evidence that the k-sum problem may even be harder than the above problems.

Two hash functions based on the group k-sum problem, SWIFFTX and FSB, were submitted to NIST as candidates for the future SHA-3 standard. Both submissions were supported by some sort of a security proof. We show that the assessment of security levels provided in the proposals is not related to the proofs included. The main claims on security are supported exclusively by considerations about available attacks. By introducing “second-order” bounds on bounds on security, we expose the limits of such an approach to provable security.

A problem with the way security is quantified does not necessarily mean a problem with security itself. Although FSB does have a history of failures, recent versions of the two above functions have resisted cryptanalytic efforts well. This evidence, as well as the several connections to more standard problems, suggests that the k-sum problem in some groups may be considered hard *on its own* and possibly lead to provable bounds on security. Complexity of the non-trivial tree algorithm is becoming a standard tool for measuring the associated hardness.

We propose modifications to the multiplicative Very Smooth Hash and derive security from multiplicative k-sums in contrast to the original reductions that related to factoring or discrete logarithms. Although the original reductions remain valid, we measure security in a new, more aggressive way. This allows us to relax the parameters and hash faster. We obtain a function that is only three times slower compared to SHA-256 and is estimated to offer at least equivalent collision resistance. The speed can be doubled by the use of a special modulus, such a modified function is supported exclusively by the hardness of multiplicative k-sums modulo a power of two.

Our efforts culminate in a new multiplicative k-sum function in finite fields that further generalizes the design of Very Smooth Hash. In contrast to the previous variants, the memory requirements of the new function are negligible. The fastest instance of the function expected to offer 128-bit collision resistance runs at 24 cycles per byte on an Intel Core i7 processor and approaches the 17.4 figure of SHA-256.

The new functions proposed in this thesis do not provably achieve a usual security property such as preimage or collision resistance from a well-established assumption. They do however enjoy unconditional provable separation of inputs that collide. Changes in input that are small with respect to a well defined measure never lead to identical output in the compression function.

Keywords: cryptography, hash functions, provable security, generalized birthday problem, knapsack cryptosystems

Acknowledgements

First and foremost I wish to thank Arjen Lenstra for the supervision during the four years I spent working in his laboratory. Thank you for all the invaluable advice and support, for the freedom and for the ideal working conditions I found here. It was the most pleasant experience imaginable.

I appreciate the many insightful comments on this thesis provided by the members of the jury. Thank you very much for all the work involved.

Several people provided me with useful feedback at various stages of my research. Special thanks in particular to Joppe Bos, Scott Contini, Dimitar Jetchev, Thorsten Kleinjung, Onur Özen, Kenny Paterson, Martijn Stam and Ron Steinfeld.

I would like to thank all my friends and colleagues from LACAL and beyond for the help I received during my stay and for the usual pleasant atmosphere I especially enjoyed at EPFL. I am happy to have worked in this community. I appreciate the help Maxime Augier provided with the French text within this thesis. Many thanks to Monique Amhof for all the administrative work as well as for the help in matters unrelated to EPFL.

I am grateful to my family for the continuous support and encouragement. This thesis is dedicated to my wife. I would not have succeeded without you.

This work was supported by a grant of the Swiss National Science Foundation number 200021-116712.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few decades, modern cryptology has become a standard security tool in the evolving digital world. It is no longer a secret military technology or an exotic academic discipline. Though often silent and invisible, cryptographic methods protect out physical and intellectual property, money and a fair share of our on-line communication. Cryptography has evolved into a broad field that can be applied in a variety of scenarios.

A *cryptographic protocol* is a method designed to achieve security in a well defined "high-level" communication scenario, such as *electronic mail*, *payments* or *voting*. Protocols normally correspond to our ordinary human needs. This thesis is devoted to cryptographic hash functions. Such objects, whatever they are, have little direct connection with everyday life. This chapter summarizes the current state of the art and provides motivation for our research.

1.1 Hash Functions and Applications

Cryptographic hash functions are one of the most basic building blocks of cryptographic schemes and protocols. Informally, such a function transforms any input message m to a short fixed-size digest $H(m)$ such that the following three properties are satisfied:

- Given a digest y, it is hard to compute a message m such that $H(m) = y$.

- Given a message m, it is hard to compute another message $m' \neq m$ such that $H(m) = H(m')$.

- It is hard to find two messages $m \neq m'$ such that $H(m) = H(m')$.

Let us mention two "standard" and widespread applications of cryptographic hash functions.

Password Authentication Virtually all computer systems use passwords to authenticate users. A computer user selects his very own secret string and shares it with the system administrator. The secret needs to be reproduced for a login attempt to be successful. Computer systems typically do not store the actual passwords, but keep a list of digests instead. On a login attempt, the password provided is hashed and the result is compared to the stored value. An attacker who obtains the list of hash values does not learn the passwords and does not gain the ability to impersonate users.

Digital Signatures Public key cryptography provides methods to sign digital messages to prove their authenticity. A valid signature should convince the recipient that the message as it was received originates from a particular source. It is standard practice that messages are hashed first and only the digest is signed. This has several advantages. The size of a signature is independent of the size of the message itself even though all the data is signed. Because real-world signing algorithms are typically much slower than hash functions, the approach allows messages to be signed faster.

1.2 Classical Hash Functions

About a decade ago, the functions MD5 and SHA-1 dominated the world of cryptographic hashing as far as deployment is concerned. The former is due to Rivest [115]. The SHA family of functions, similar in design, was established as a standard by NIST [99]. A superficial look at the internal design of the functions reveals dozens of rounds performing a varied mix of simple operations on binary words and magic constants. An attacker is likely to be lost in the web of computations performed on message pieces. The functions are *complex* in design.

This is one of the features that kept them secure. Security was assessed exclusively by means of attacks. Both functions fared reasonably well in this respect and suffered a few non-critical drops in security. The breakthrough results of Wang et al. led to powerful attacks on both the functions [149, 148].

As a result of the developments, collisions in MD5 can now be computed almost instantly on an ordinary workstation. Stevens et al. used optimized collision-finding methods to create a rogue Certificate Authority [137]. Such an attack, had it been executed by a team with malicious intent, could have severely compromised then-existing security infrastructure of the Internet.

As of October 2011, an actual pair of colliding inputs to SHA-1 is yet to appear, the complexity of collision search is estimated to be roughly 2^{60} steps.

1.3 The Future Standard

The SHA family includes more recent functions[1] that appear to be secure for the time being. Because they are built on similar principles as SHA-1, NIST had identified the need to establish a new standard to be called SHA-3. It will be selected in a public competition that opened in late 2008 and is now underway [100]. Over sixty proposals were originally submitted. The process is now in its third and final round with five finalists. The winner shall be announced in 2012. Two now eliminated submissions to the competition are included in this thesis.

1.4 Hash Functions and Provable Security

The cryptanalytic results by Wang motivated more research in the area of hashing and the efforts have not been limited to the competition run by NIST. The wide-spread functions such as MD5 are sometimes called *ad-hoc* or *dedicated* hash functions, they are built directly from scratch without using an established lower-level cryptographic primitive or method.

"Provable security" is a principle that governs cryptographic design in order to identify, define and finally prove security properties. It promotes the use of established simple primitives to build more sophisticated constructions. If the simple primitive is known or believed to be secure, it can be used to show security of the higher-level construction. It appears to be impossible to prove security of *all* cryptographic primitives from the security of other cryptographic primitives without making some assumptions. One has to start somewhere and lay out the foundations first.

Hash Functions from Block Ciphers A natural ingredient to look at is a *block cipher*, a symmetric primitive designed for encrypting data in blocks

[1] SHA-2

under a single (secret) key. Hash functions based on block ciphers are studied extensively and often enjoy well established provable security properties [113, 25, 133]. Secure block ciphers lead to secure hash functions. The arguments why the modern block ciphers are fit for this purpose may be considered reminiscent of the arguments why the SHA family was thought to be secure only a few years ago. On the positive side, the approach benefits from the speed of block ciphers and delivers functions able to meet the usual requirements on performance. Hash function design is simplified considerably by outsourcing some of the design effort. One ends up with a *simple* hash function secure under a rather *complex* assumption.

Hash Functions and Public-Key Cryptology Public-key cryptology is a fascinating branch of cryptology that builds encryption and signature schemes from assumptions on *hard* problems formulated as *simple* mathematical statements. It was known since the beginning of modern cryptology that the very same problems can be used to build hash functions. These were unfortunately terribly slow in comparison to dedicated functions and were mainly of academic interest. The fall of MD5 and SHA has brought motivation to re-consider them. This thesis is devoted to hash functions from assumption related to public-key schemes. We shall argue such functions are worth looking at, for they are reasonably *fast*, *simple* in design and often can be proved secure under a *simple* assumption.

1.5 Outline of the Thesis

Chapter 2 provides the basic definitions of hash functions and our expectations on security. A selection of more or less common computational problems useful in public-key cryptology appears in Chapter 3 along with the perceived hardness and their applications. We also sketch the relevant algorithms for solving the problems. Chapters 4 and 5 survey several existing functions with surprisingly similar structure and stress their common features. This systematic study provides context and justification for main contributions of this thesis in the final two chapters. There we propose new families of hash functions that operate in multiplicative finite groups. Our designs achieve performance comparable to the current standards and their security is related to a computational problem that has been around for decades.

Chapter 2

Hash Function Basics

This chapter is devoted to definitions of the basic properties and security of cryptographic hash functions. There appears to be no single ideal way to do so. Design goals, security definitions and expectations evolve. Proposals of the functions considered in this thesis stretch over a time span no shorter than 30 years. In order to be able to interpret all the variety within a single coherent picture, we will develop a general approach to security assessment. Parts of this chapter were published in the article *Interpreting Hash Function Security Proofs* [122].

2.1 Definitions

A *hash function* is a mapping of the form

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

for an integer n. Such a function takes an arbitrary string of bits as input and produces a string of bits of fixed length n. If the domain of H is restricted to $\{0,1\}^l$ for an integer $l > n$, then H is called a *compression function.*

We require H to be efficiently computable. We shall look for properties that can make such functions useful in the cryptographic context.

2.1.1 Random Oracle Methodology

Protocol designers may use a hash function as a real-life replacement of a fixed-length *random oracle.* Such an oracle can be queried with any string from $\{0,1\}^*$ and outputs n random bits. Outputs for different queries are independent, but whenever a query is repeated the value returned is the same.

If a protocol is proved secure provided access to a true random oracle, it may be considered secure if the oracle is replaced by a hash function computation in reality. The paradigm was suggested by Bellare and Rogaway [18] as a justification for the use of hash functions in protocol design. Canetti et al. later showed that one can craft schemes and prove them secure if a random oracle is used, yet the schemes are insecure with any real-world replacement thereof [32]. Bellare et al. also exposed a scheme that becomes insecure if random oracles are replaced by a real implementation [16]. The random oracle methodology was defended by Koblitz and Menezes who described the counterexamples as contrived [75]. If we stick to the methodology, we need a reasonable real-world instantiation of the oracle. Bellare and Rogaway proposed schemes that used then available hash functions MD5 and SHA. The limitations of such constructions were shown by Leurent and Nguyen [82]. Following the attacks on the hash functions, we may want to update the arguments and use current improved successors of MD5 and SHA that are better suited as replacements of random oracles. A reasonable successor has not been established yet. We may decide not to follow the random oracle methodology and build cryptographic schemes on objects that are, hopefully, easier to obtain.

2.1.2 Hash Function Security Properties

The "standard" security properties are the three we listed in Section 1.1, called *preimage resistance*, *second preimage resistance* and *collision resistance*, respectively [89].

Rogaway and Shrimpton studied several formal definitions of these properties and their relationship [118]. We adapt the formalism to better suit our needs. In particular, we separate the task of the adversary from the cost involved.

The goal is to find a single fixed hash function H for a particular length of interest n. For technical reasons it may be more appropriate to work with larger families of functions and then select H as a single member of the family. Many constructions, in particular within this thesis, do inherently come with parameters that can be varied. Consider H to be a member of a family of functions $\mathcal{H}_{\mathcal{K}}$ for a set $\mathcal{K}$. An element of $\mathcal{K}$ shall be called a *key*, every $K \in \mathcal{K}$ corresponds to a single function H_K. When a single function H is selected from the family $\mathcal{H}_{\mathcal{K}}$, the particular choice of K is made public. This generalization includes unkeyed functions, simply consider $|\mathcal{K}| = 1$.

Let $\mathcal{H}_{\mathcal{K}}$ be a family of hash or compression functions. Consider adversaries facing one of the following tasks:

Preimage search Given a random key $K \in \mathcal{K}$ and a random element y from the range of H_K find m such that $H_K(m) = y$.

Second Preimage search Given a random key $K \in \mathcal{K}$ and a random message m find $m' \neq m$ such that $H_K(m) = H_K(m')$.

Collision search Given a random key $K \in \mathcal{K}$ find two messages m and m' such that $m \neq m'$ and $H_K(m) = H_K(m')$.

For a family $\mathcal{H}_\mathcal{K}$ to be considered "secure" we expect the above problems to be hard. In other words, there shall be no adversary that would have an easy job with one of the three tasks. To be able to reason about hardness, we need a way to assign cost to adversaries. There are several ways to do so. For the moment we shall work with a general abstract framework that stresses our main requirement on a cost, that is the ability to *compare*.

2.1.3 Modeling cost

Let a hash function family $\mathcal{H}_\mathcal{K}$ be fixed. Let levels of computational cost be represented by elements of a non-empty set S. The precise nature of S will vary between families of functions. In order to allow different approaches to cost, we only require S to be a partially ordered set, i.e. equipped with a *reflexive, antisymmetric* and *transitive* relation '$\leq$'. While it might appear natural to require the ordering on S to be *linear* as well, it is not necessary for our purposes. This extra freedom may even be desirable. Let the set S contain a least element s_0 and a greatest element s_∞.

So far we can compare cost levels. Our second expectation is the ability to *scale* by a positive constant. Allow multiplication of $s \in S$ by a positive real number c such that $s \leq cs$ for $c \geq 1$. We require $c(ds) = (cd)s$ and that $s \leq t$ implies $cs \leq ct$ for constant c, d. Multiplication of cost levels then also satisfies $1s = s$ for all $s \in S$, i.e. it is an action of the multiplicative group $\mathbb{R}_+^*$ on S.

Cost functions

Let $\mathcal{H}_\mathcal{K}$ be a fixed family of hash functions, define $\mathcal{A}_{\text{Pre}}$ to be the set of all probabilistic algorithms that read the description of a preimage search problem and possibly output a solution. Define the sets $\mathcal{A}_{\text{Sec}}$ and $\mathcal{A}_{\text{Col}}$ to represent adversaries solving the second preimage and collision search problems, respectively. Define a cost function $c : \mathcal{A}_{\text{Pre}} \cup \mathcal{A}_{\text{Sec}} \cup \mathcal{A}_{\text{Col}} \to S$ that assigns cost to adversaries. Although adversaries as well as the corresponding challenges are in general probabilistic, we require the cost of an adversary A to be represented by a single object $c(A) = s \in S$. This is only a matter of notation, the abstract nature of cost permits s itself to be a random variable or a more sophisticated object.

Our approach to cost is *very* general. For example, the set S can only contain a single element s, i.e. there is only a single level of cost. In such a case all multiplication by a positive constant collapses to the identity mapping, $cs = s$ for all $c > 0$. All attacks in this setting cost the same.

As the other extreme, the cost of any two algorithms can be incomparable. This happens if c is an injective mapping into a subset of isolated points of S. Multiplication can be defined such that $cs \neq s$ for $c \neq 1$, yet no multiple of $c(A)$ can be compared to any multiple of $c(B)$ for $A \neq B$.

We propose a *lazy* approach to defining S, c and in particular the ordering $\leq$. Only define the elements and relations when needed. The use of scaling allows this to be done in a cautious way. Given two levels $s, t \in S$, we may suspect the two are close to each other, yet be unable to determine whether $s \leq t$ or $t \leq s$. It is possible to have e.g. $s \leq 100t$ and $t \leq 100s$ independent of the relationship of s vs t.

To Attack or to Prove? Intuitively, security of a scheme would be equal to the cost of the most efficient attack imaginable. There are two principal ways to approach such a quantity. One can certainly imagine attacks one at a time and relax the security claims when a faster attack is discovered. Essentially, a scheme is presumed secure until proven otherwise.

On the other hand, the *provable security* approach tries to deal with *all* the attacks at once and provide an argument why no attack can succeed.

The two approaches should be viewed as complementary rather than opposing. We shall consider security to be an *a priori unknown* estimate on the cost of attacks.

Bound Types

Let $\mathcal{A}$ represent either of $\mathcal{A}_{\text{Pre}}$, $\mathcal{A}_{\text{Sec}}$ or $\mathcal{A}_{\text{Col}}$. Define the following two main *types* of bounds on security:

- $p \in S$ is a type **L** bound if for every adversary $A \in \mathcal{A}$ the cost $c(A)$ is greater than or equal to p.

- $q \in S$ is a type **U** bound if every type **L** bound $p \in S$ is less than or equal to q.

A type **L** bound s by definition provides a lower bound on the cost of all attacks. We intend to define security as a measure of hardness, we shall therefore call s a *lower bound on security.*

A type **U** bound is defined as an upper bound on all type **L** bounds. By definition, the cost level $c(A)$ is a type **U** bound for all adversaries A. Call any type **U** bound q an *upper bound on security.* This corresponds with the intuition that hardness of a problem can never exceed the cost of an algorithm solving it.

Observe that our definitions allow $q \in S$ to be a type **U** bound even if there is no $A \in \mathcal{A}$ such that $q = c(A)$. It is also possible that $p \in S$ has type **L** and there is no $A \in \mathcal{A}$ such that $p = c(A)$. If p is has type **L** and q has type **U**, then $p \leq q$.

We shall call a family $\mathcal{H}_{\mathcal{K}}$ *provably secure* with respect to $\mathcal{A}$ if we prove that $p \in S$ is of type **L**. The value p will then be the provable security level. In the context of hash function preimages and collisions, we will also use the term *resistance.*

Our approach to provable security does not explicitly prescribe the number of possible keys $|\mathcal{K}|$. However, if $|\mathcal{K}| = 1$ and the family only contains a single function, there exists a trivial collision-finding adversary A. The adversary simply outputs two hard-coded messages m and m'. The cost associated to this adversary has type **U**, therefore any **L** bound is less than or equal to $c(A)$. Instead of trying to design a cost model where the level $c(A)$ provides enough confidence, use large $\mathcal{K}$ to increase cost of such adversaries. In practice the key K will eventually be fixed and attacks specific to the fixed function may exist. A security proof valid for random H_K no longer rules out all the attacks imaginable. We heuristically expect that it prevents *most* of them, provided the parametrization of a family was not put in place only as a work-around to provide "provable security" for an otherwise fixed hash function. The scenario as well as ways to build provable security arguments on un-keyed hash functions were discussed by Rogaway [117].

A statement on the cost of a single attack needs to provide no information on the relative cost of any other attack. For a type **U** bound q, there may exist any combination of attacks that cost strictly more or strictly less than q. On the other hand, an **L** bound is a statement on the cost of *all* attacks. We assume that whenever a provable security claim is made, a sufficiently rich ordering of cost levels is already in place.

Bounds on Bounds Because type **U** bounds are tied to attacks, they are quite common. Designers do not often establish bounds of type **L**. To interpret certain results, we will need the two following auxiliary bound types:

- $r \in S$ is a type **lU** bound if $r \le t$ for t of type **U**. This means r is a lower bound on *some* upper bound on security.

- $s \in S$ is a type **uL** bound if $s \ge t$ for t of type **L**, i. e. s is an upper bound on *some* lower bound on security.

Elementary Properties of Bounds Every **L** bound is an **lU** bound and every **U** bound is an **uL** bound. Such implications are trivial, as *any* element of S has type **lU** and **uL** simultaneously relative to s_0 and s_∞. The "weak" types **uL** and **lU** apparently provide no direct useful information on provable security. Both are related to a bound t of one of the "useful" types **U** and **L**. The types **uL** and **lU** can nevertheless carry valuable partial information about the bound t if it is not quantified otherwise. For example, an **uL** bound tied to a particular reduction limits how good the reduction is and an **lU** bound tied to a particular attack algorithm limits how much damage the attack causes. We will see such examples later.

2.1.4 Examples

There are several ways to measure cost of attacks on a cryptographic primitive. Practical examples include hardware cost, processor cycle count, memory or simply real time taken. On the more theoretical side, one can measure success probability as a function of runtime, expected runtime, circuit size, etc. The choices of S are typically tailored to match a particular hardness assumption or a proof method. Sometimes the elements of S will turn out to be numbers counting some basic operation, such as a bit operation or evaluation of H_K.

Asymptotic Security In complexity theory, *polynomial time* algorithms are understood to be efficient and problems that cannot be solved in polynomial time are considered hard. The same classification may be used to define security of cryptographic schemes. An infinite family of hash functions indexed by a parameter n would be called secure if no adversary bounded by a polynomial in n can succeed in violating the desired security property for $n \to \infty$. This can be reflected in our framework by a set S containing only two elements: $s_0 \le s_\infty$. The former represents polynomial-time algorithms and the latter represents the rest. Multiplication by a constant leaves both elements of S unchanged. If s_∞ is shown to be a type $\mathbf{L}$ bound, then the family is provably secure. Security in this setting is not quantified otherwise. It is a yes/no property rather than a measure. There is no room for a finer scale in the two-valued cost model. The language of such provable security arguments is incompatible with the view of a practitioner. Many asymptotic security arguments can be adapted to quantify security also for finite values of n where needed. Koblitz and Menezes pointed out several schemes that are provably secure in the asymptotic sense but fail to provide any security guarantee for instances of practical interest [74].

Concrete Security Cryptanalysts normally speak of rather precise time estimates for programs running on real world hardware. In order to allow proofs and attack to complement one another, both should use similar language. In other words, proofs need to provide lower bounds on the effort needed to break a function in the same units. In this setting, security is not a property, but a measure. It needs to be quantified and made as exact as possible. The concrete approach was advocated by Bellare [15].

2.2 Towards Proofs of Security

A security proof sets a lower bound on all attacks on a particular security property. It is usually a *conditional* statement relying on some hardness assumption. Confidence is transferred from the hard problem to the hash function by means of a reduction. It shows that any algorithm that succeeds in breaking our cryptographic scheme can be adapted for solving an instance of a problem P that is presumed hard. Such an argument will connect the hardness of the problem P to the cost of attacks.

Any hash function family can be made provably secure in the above sense for an appropriate choice of the hardness assumption. In order for a reduction to be of any use at all, there has to be some confidence in the assumption to begin with. We will list some “standard” assumptions in Chapter 3 and point out a few unfortunate choices in Chapter 5.

The importance of relativity in provable security arguments cannot be overstated. Strictly speaking, the term *provable security* is misleading. Bellare made a suggestion to use *reductionist security* instead [15], followed by Koblitz and Menezes [75].

2.3 Generic Attacks

Given the value y one can mount a brute-force preimage attack by hashing random messages m until $H(m) = y$. The expected number of trials needed is 2^n. This straightforward attack requires little memory and can easily be parallelized. It provides a generic **U** bound on preimage and second preimage resistance of hash functions.

The same approach can be used to find collisions by searching for two preimages of a randomly selected value, but due to the well-known birthday paradox one can do significantly better. Start hashing random messages m and keep track of all the candidates until two of them collide under H. The expected number of trials until a collision is found is approximately $\sqrt{\pi/2} \cdot 2^{n/2}$. The high memory requirements of the attack can be avoided with the use of appropriate cycle-finding techniques. Van Oorschot and Wiener described a variant that can be parallelized efficiently [142]. In all cases, the number of evaluations of H needed is approximately $2^{n/2}$.

The two generic attacks have traditionally served as benchmarks of hash function security. Some definitions require the associated type **U** bounds to be achieved, it is in particular expected of the future SHA-3 [100]. We view this as quite an ambitious goal. Functions with less-than-ideal security need not be immediately discarded. It may turn out that reasonable **L** bounds are achievable.

2.4 MQ-HASH - An Example

We apply the classification of bound types to interpret various provable security results throughout this thesis. As an example, we take a detailed look on the security arguments behind a compression function proposal MQ-HASH. It is a function introduced by Billet et al. built on the hardness of solving systems of multivariate quadratic equations over $\mathbb{F}_2$ [24]. The function was claimed to establish provable preimage resistance. It is inspired by the stream cipher QUAD by Berbain et al. [19]. Provable security arguments supporting QUAD were questioned by Yang et al. [150]. Aumasson and Meier cryptanalyzed MQ-HASH, but did not attack the provable property [10].

Definition

The compression function maps $n + k$ bit input to n bits. Let $r \geq n + k$, and f be an r-tuple of quadratic polynomials in $n + k$ variables over $\mathbb{F}_2$.

Let g be an n-tuple of quadratic polynomials in r variables over the same field. The precise values proposed are $k = 32$, $n = 160$ and $r = 464$. This means f maps 192 bits to 464 bits and g maps 464 bits to 160 bits. Both f and g are to be selected uniformly at random. The compression function of MQ-HASH is the composition of f and g mapping 192 bits to 160 bits.

Hardness Assumption

The designers derive the security from the hardness of solving random systems of multivariate quadratic equations over the binary field. This is a special case of the NP-Complete MQ problem [53].

It is assumed that solving systems of the type of f and g is hard. For f the hardness is quantified at $l_f = 2^{103.88}$ binary operations. The bound follows from the analysis of a particular solving algorithm performed by Bardet et al. [11]. Billet et al. assume no faster attack exists, the quantity l_f is therefore a lower bound on the cost of inverting systems such as f. The implicitly present lower bound on the cost of inverting g is not quantified. Denote it by l_g.

Security Claims

The designers claim the function is preimage resistant by proving the following Theorem.

Theorem 2.1. *Let T_f and T_g denote the time required to evaluate f, resp. g. Let A be an algorithm inverting (a random) $g \circ f$ in time T with probability ε. Then A can be either converted to an algorithm inverting g in time $T + T_f + T_g$ with probability ε or to an algorithm that can invert randomly chosen tuples of 464 quadratic polynomials in 192 variables that runs in time*

$$T' = \frac{128 \times 192^2}{\varepsilon^2} \left(T + T_f + 3T_g + \log \left(\frac{128 \times 192}{\varepsilon^2} \right) + 464 \times 192 + 2 \right) \tag{2.1}$$

and succeeds with probability $\varepsilon/2$.

The theorem does indeed establish a lower bound on cost of any algorithm inverting $g \circ f$ under the hardness assumption. It is not immediate from the statement how security can be quantified.

Suppose that A is an algorithm that can invert a MQ-HASH instance $g \circ f$ in time T. If A can be converted to an algorithm that inverts g, then $T + T_f + T_g \geq l_g$. This implies a lower bound $l_1 = l_g - T_f - T_g$ on T.

On the other hand, if A can be converted to an algorithm that inverts random systems f, then $T' \geq l_f$ by the assumption on the cost of solving such systems. By Equation (2.1) and because $\varepsilon \leq 1$, conclude that

$$T \leq \frac{T'}{128 \times 192^2} \ . \tag{2.2}$$

If Equation (2.1) translates the lower bound l_f on T' to a lower bound $l_2 \leq T$, then inequality (2.2) implies

$$l_2 \leq \frac{l_f}{128 \times 192^2} .$$

Substitute the known value $l_f = 2^{103.88}$ in the above inequality to obtain $l_2 \leq 2^{82}$. In other words, the *lower* bound l_2 is *less than* or equal to 2^{82}.

Both the alternatives in Theorem 2.1 lead to lower bounds on T, either $T \geq l_1$ or $T \geq l_2$. These in turn result in a type **L** bound l on preimage resistance that satisfies $l \leq l_1$ and $l \leq l_2$. Because l_g is not quantified, neither is l_1. We can nevertheless conclude that $l \leq l_2$.

Although the theorem does imply a type **L** bound l, Billet et al. do not quantify it. The available information allows us to find an upper bound on l. Theorem 2.1 establishes that 2^{82} is a bound of type **uL** on preimage resistance.

The designers of MQ-HASH aim at "80-bit security" and claim the level is consistent with the implications of the Theorem. Although the inequality $2^{80} \leq 2^{82}$ certainly holds, the latter quantity is merely an upper bound on l_2. The actual value of l_2 may even be less than 2^{80}. A cleaner connection of T and T' may improve the present 2^{82} bound. In order to derive an **L** bound, a quantified l_g needs to be filled in. Without the value, the original security assumption is incomplete.

The quantity 2^{82} counts bit operations. If we want to translate this to the equivalent of hash function computations, divide by the cost of such an evaluation, estimated to be 2^{24} bit operations. The authors do not comment on how the compression function is to be computed, but their 3 MB memory requirement is consistent with our estimate. The **uL** bound on preimage resistance then becomes $2^{82}/2^{24} = 2^{58}$ evaluations of the compression function.

Conclusion The security proof does not rule out preimage attacks that cost the equivalent of 2^{58} evaluations of MQ-HASH. We do *not* claim such an attack exists, our statement only exposes the limits of the security guarantees the proof can provide. Improved proofs may be possible.

2.5 Domain Extenders

Merkle and Damgård independently described a method that allows to turn a compression function $h : \{0,1\}^l \to \{0,1\}^n$ to a hash function H processing long messages [91, 44]. The construction is commonly referred to as the Merkle-Damgård mode. Fix an n-bit constant initialization vector IV, let the integer $b = l - n$ represent block length and select a positive integer $c < b$. To hash an $\mathcal{L}$-bit message m for $\mathcal{L} < c$ proceed as follows:

1. Initialize $x_1 = IV$.
2. Append a single 1 bit to the message, then pad with zero bits until the total length is c bits short of an integral multiple of b.
3. Append a c-bit binary representation of the length $\mathcal{L}$, let k bet the number of b-bit blocks in the string obtained.
4. For $i = 1, \ldots, k$ compute $x_{i+1} = h(x_i || m[i])$.
5. Return x_{k+1}.

The operator $||$ denotes concatenation of binary strings and $m[i]$ is the i-th b-bit block of the bk-bit pre-processed message obtained in Step 3. The above construction extends the input domain of h from l bits to $2^c - 1$ bits, this can be made big enough for all practical purposes. Finding collisions in H is no easier than in h, any collision in the compression function trivially extends to H.

Theorem 2.2 (Merkle, Damgård)**.** *If $m \neq m'$ collide under H, then a collision in h must also have occurred in the course of the above algorithm.*

Proof. If m and m' differ in length, the representations of length appended to the messages differ, i.e. $m[k] \neq m'[k']$. We immediately obtain a collision in the last call to h with the inputs $x_k || m[k]$ and $x'_{k'} || m'[k']$. It remains to prove the theorem for $|m| = |m'|$. If there exists an index $2 \leq i \leq k - 1$ such that $x_i \neq x'_i$ and $x_{i+1} = x'_{i+1}$, then $h(x_i || m[i])$ collides with $h(x'_i || m'[i])$. Otherwise $x_i = x'_i$ for all $1 \leq i \leq k$. Let $1 \leq j \leq k$ be an index such that $m[j] \neq m'[j]$, it exists because the inputs differ. The string $x_j || m[j]$ collides with $x'_j || m'[j]$. □

Drawbacks of Iterated Constructions A hash function H that uses the Merkle-Damgård iteration is as secure with respect to collision search as h. Joux showed how to create *multicollisions* by combining a small number of collisions in h. Due to the iterative structure used, t appropriately chosen pairs of colliding blocks lead to 2^t messages t-block long all with the same digest [65]. Kelsey and Schneier applied the ideas to find second preimages in much less than 2^n evaluations of h [72]. Kelsey and Kohno extended the results of Joux and developed a preimage attack as well [71]. All the above attacks are generic in the sense that they allow substantially faster preimage search in the iteration even if the building block h achieves maximum possible security.

There are new modes of operation designed to prevent the attacks, for example HAIFA by Biham and Dunkelman [23]. For the purposes of this thesis, whenever a compression function is considered, we assume a suitable iterated mode of operation is available to turn it to a hash function.

Chapter 3

Hard Computational Problems

This chapter lists some of the best known computational problems used in modern public-key cryptology that we will refer to in the later chapters. Several of the problems listed are today considered “standard” and support schemes widely deployed in practice. For every problem considered we list the relevant theoretical lower bounds and solving algorithms. We include the related public-key cryptographic schemes. There is a compression function based on every problem considered, these are postponed to the later chapters.

3.1 Integer Factorization

Problem 3.1 (Integer Factorization)**.** *Given a positive integer M, find the unique integer $u > 0$, prime numbers p_i and positive integers β_i for $i = 1, \ldots, u$ such that*

$$M = \prod_{i=1}^{u} {p_i}^{\beta_i} \ .$$

Assume that the integer M is composite. Although we are interested in a complete factorization, it is sufficient to look for algorithms that express M as a product of two smaller positive integers. The task can then be completed recursively.

3.1.1 Algorithms

To express the cost of factoring algorithms, we shall use the notation

$$L_x[\alpha, c] = e^{(c+o(1))(\log x)^{\alpha}(\log\log x)^{1-\alpha}},$$

for $0 < \alpha < 1$ and $x \to \infty$, the logarithms are natural. Algorithms with such complexity are called *subexponential* in $\log x$. The dominant parameter is α, the smaller the better.

Many instances of the integer factoring problem are easy. It can be solved efficiently for example if M is divisible by no more than a single "large" prime. There are algorithms with runtime that depends on the *smallest* of the prime divisors of M, even trial division will find some factor quickly for most random M. The ECM method by Lenstra runs in time $L_p[1/2, \sqrt{2}] \cdot O\left((\log M)^2\right)$ for p the smallest prime dividing M [81]. It is common to call a number *hard to factor* if it is a product of two prime numbers of roughly the same size.

The most efficient general-purpose factoring algorithms are sophisticated extensions of an old method by Fermat. If $M = X^2 - Y^2$ for positive integers X, Y, then $M = (X+Y)(X-Y)$ is a non-trivial factorization of M provided $|X - Y| \neq 1$. The task can be relaxed to finding integers such that $X^2 - Y^2 = 0 \mod M$. One can then efficiently check for common divisors of M and $X + Y$. This will split M with at least 50% chance for random X, Y. We will briefly sketch the modern methods for finding useful X, Y. In order to describe the inner workings of modern factoring algorithms, we will need the notion of *smoothness.*

Definition 3.1. *An integer is called B-*smooth *if it has no prime divisors greater than B.*

Denote the i-th prime number by p_i and let $p_0 = -1$. Let u be the index of the largest prime number not exceeding B. We will sketch two well-known modern methods for factoring integers.

Quadratic Sieve

The following two-stage approach originates in the work of Morrison and Brillhart [98]. First collect expressions of the form

$$\prod_{i=0}^{u} p_i^{e_i} = v^2 \mod M \tag{3.1}$$

for integer v such that not all e_i are even. Such an expression is called a *relation* and $(e_0, \ldots, e_u)$ an *exponent vector*. The set of primes p_0 to p_u is a *smoothness basis*. A relation corresponds to an integer v with a square that factors over the basis. After collecting at least $u+2$ relations, proceed to the second stage to find a linear dependence modulo 2 among the corresponding exponent vectors. This is done by linear algebra on a $0-1$ matrix of the reduced exponent vectors. Given a solution, multiply the relations accordingly to obtain X, Y such that $X^2 \equiv Y^2 \mod M$. The optimal choice of B depends on several factors, in particular on the distribution of B-smooth integers. For small B, relations are scarce. For increasing B, a single relation gets easier to find, but the number of relations needed grows. With optimal parameters, the algorithm runs in expected heuristic time $L_M[1/2, 1]$.

Number Field Sieve

Currently the fastest known general purpose factoring algorithm originates in the work of Pollard improving the Morrison-Brillhart approach [112]. The smoothness basis is extended by prime ideals of a suitable algebraic number field. The right hand side of relations collected is formed by products of algebraic primes. The extra freedom allows to use significantly lower smoothness bounds and leads to complexity $L_M[1/3, (64/9)^{1/3}]$ [77].

Computational Experiments As of 2011, the largest general number publicly reported as factored by NFS was a 768-bit product of two primes. This was achieved in 2009 by Kleinjung et al. in a coordinated effort equivalent to some 2000 years of computation on a single core of a then-common CPU [73]. The next challenge, a 1024-bit modulus was estimated to be about a thousand times harder.

3.1.2 Related Problems

The following two problems are closely linked to the factorization problem. The ability to solve any of them efficiently allows to factor M and vice versa.

Modular Square Root To factor M, generate a random X and obtain a modular square root Y of X^2. This leads to a congruence of squares $X^2 = Y^2$ that is useful at least half of the time.

Computing the Order of $\mathbb{Z}_M^*$ The multiplicative group of integers modulo M has precisely $\varphi(M)$ elements, where φ stands for Euler's function.

The quantity can easily be computed from the factors of M. There exists an efficient probabilistic algorithm that splits M given $\varphi(M)$. If M is a product of two primes, the factors can be recovered from $\varphi(M)$ by solving a simple quadratic equation.

3.1.3 Related Cryptosystems

A public-key cryptosystem proposed by Rabin encrypts messages by squaring them modulo a product of two primes [114] and is therefore secure if factoring is hard.

RSA by Rivest et al. is probably the most widely deployed public-key system [116]. Messages are encrypted by raising to a fixed power e modulo a product of two primes. It can be broken by computing e-th modular roots. It is believed that the fastest method to do so for general e is by factoring M. The knowledge of $\varphi(M)$ then allows to compute a decryption exponent d such that $ed = 1 \mod \varphi(M)$.

3.2 Discrete Logarithm

Every element of a cyclic group is an integral power of the group generator. We are concerned with the problem of recovering the exponent given the power.

Problem 3.2 (Discrete Logarithm (DL))**.** *Let G be a finite group generated by g. Given an element $y \in G$, find the unique exponent $0 \leq x \leq |G| - 1$ such that*

$$g^x = y \ .$$

3.2.1 Algorithms

It is easy to find groups where the problem is trivial, such as $(\mathbb{Z}_M, +)$. Discrete logarithm is a very general problem, a proper choice of G is crucial.

Generic Algorithms

If the order of $|G|$ is composite and the factors are known, the logarithm of y can be recovered from logarithms in prime-order subgroups by means of an algorithm by Pohlig and Hellman [109]. The Rho method by Pollard recovers x in approximately $\sqrt{|G|}$ steps of computation [111]. In order to make the discrete logarithm hard, one often selects G of prime order, or an order divisible by a large prime.

By combining the two algorithms one obtains a generic solving algorithm with complexity $O(\sqrt{p})$ for p the greatest divisor of $|G|$. Shoup showed that $\Omega(\sqrt{p})$ group operations are needed in any generic group [131]. Much faster algorithms are available for several choices of G, in particular for multiplicative groups of finite fields.

Index Calculus in Finite Fields

Logarithms in multiplicative groups of finite fields can be computed by a three stage approach that shares certain features with fast factoring methods from the previous section. First select an appropriate smoothness basis P and collect multiplicative relations among the elements. Such relations translate to additive relations of logarithms.

After collecting enough relations, compute logarithms of all the elements of P using linear algebra. Computations are performed modulo the group order or a divisor thereof.

The third stage allows computation of logarithms for individual field elements. Given an arbitrary $y \in \mathbb{F}$, find a multiplicative relation in g, y and the elements of P. Then recover $\log_g y$ from the additive relation of the corresponding logarithms.

There are several index calculus algorithms each suited for a different class of finite fields. Gordon considered a variant of Number Field Sieve in prime fields [58]. Adleman and Huang developed the Function Field Sieve to solve logarithms efficiently in fields of small characteristic and large extension degree [1]. Various versions of the two approaches were recently considered by Joux et al. [67, 68]. As a result of the many developments, there are $L_q[1/3, c]$ algorithms for computing discrete logarithms in any q-element finite field.

Computational Experiments The scale of experiments computing logarithms in multiplicative groups of finite fields does not compare to the efforts in the integer factoring area. Real-world hardness estimates need to be extrapolated from data on much smaller instances. By the similarity of the solving algorithms, reasonable estimates can be made from experiments factoring integers with NFS.

Elliptic Curve Groups

The fastest known algorithm to compute discrete logarithms in a group of points on an appropriately selected elliptic curve over a finite field is the generic $\Omega(\sqrt{|G|})$ Pollard Rho method. In other words, such groups are at the moment believed to behave like generic groups [39].

3.2.2 Related Cryptosystems

Public-key schemes with security connected to the hardness of discrete logarithms include ElGamal encryption and signature schemes [52] and the key establishment protocol by Diffie and Hellman [45]. These are related to the problem of computing g^{ab} given g^a and g^b known as the Diffie-Hellman problem. Variants of both systems are widely deployed.

3.3 Integer Lattices

Let $d > 1$ be an integer, let $\mathcal{B} \subseteq \mathbb{Z}^d$ be a set of d linearly independent vectors $b_1, \ldots, b_d$. The set

$$\mathcal{L} = \{c_1 b_1 + \ldots + c_d b_d\}$$

for integer c_i is a full rank d-dimensional lattice. Let $||\cdot||$ be a norm in $\mathbb{Z}^d$ assumed to be the l_2 norm if not specified otherwise. Consider the following two computational problems:

Problem 3.3 (Shortest Vector Problem, SVP)**.** *Given a lattice basis $\mathcal{B}$, find a vector $x \in \mathcal{L}$, $x \neq 0$ such that $||x||$ is minimal.*

Problem 3.4 (Closest Vector Problem, CVP)**.** *Given a lattice basis $\mathcal{B}$ and a vector $y \in \mathbb{R}^d$ find a vector $x \in \mathcal{L}$ such that $||y - x||$ is minimal.*

Both problems can be considered in approximate versions, where the solution is required to be within $\gamma(d)$ from the minimum.

3.3.1 Lower Bounds

Ajtai showed that exact SVP is NP-hard in the l_2 norm under randomized reductions [3]. Micciancio extended the result to the approximate SVP for $\gamma \leq \sqrt[p]{d}$ in the l_p norm for all $p \geq 1$ [92]. Van Emde Boas proved the exact CVP problem NP-hard in l_p for all $p \geq 1$ [141].

3.3.2 Algorithms

The LLL basis reduction algorithm by Lenstra et al. is a polynomial-time algorithm that solves SVP within exponential approximation factors [78]. Schnorr proposed a family of algorithms with smaller, but ultimately exponential, approximation factors [124]. Practical improved variants appeared in [126, 125].

Computational Experiments Nguyen and Gama analyzed performance of various lattice reduction algorithms on a range of random lattices [51]. Variants of SVP were reported to be easy to solve exactly in dimensions below 70. Only approximation had a chance beyond dimension 100, general lattices with $d = 500$ can be considered out of reach.

3.3.3 Related Problems

The NP-hardness arguments provide evidence that the two problems are likely to be hard in the worst case. Ajtai showed how to construct a class of lattice problems that are hard to solve *on average* if SVP is hard to approximate *in the worst case* [2, 4]. The definition of the class of lattices depends on two absolute constants C_1 and C_2 and parameters k, q such that $k = [C_1 d \log d]$ and $q = [d^{C_2}]$. Let A be a $d \times k$ matrix over $\mathbb{Z}_q$. The set of vectors $x \in \mathbb{Z}^k$ such that $Ax = 0 \mod q$ is a lattice, denote it by Λ. If there is an algorithm that can find a short vector $||x|| < d$ in Λ for random A in polynomial time, there is a polynomial time algorithm that solves approximate SVP in *all* d-dimensional lattices.

3.3.4 Related Cryptosystems

Goldreich et al. proposed an encryption and signature schemes with security linked to lattice reduction [57], the system was broken for most proposed parameter choices by Nguyen [101]. Ajtai and Dwork proposed a public-key encryption scheme with worst case / average case equivalence [5].

3.4 Subset Sum

Problem 3.5 (Subset Sum)**.** *Let $a_1, \ldots, a_k$ be positive integers. Given an integer y, find $I \subseteq \{1, \ldots, k\}$ such that*

$$\sum_{i \in I} a_i = y \ .$$

A solution corresponds to a vector $m \in \{0,1\}^k$ such that $\sum_{i=1}^{k} m_i a_i = y$. We sometimes use the term *knapsack* to describe the problem. Subset sum was shown NP-Complete by Karp [70].

3.4.1 Related Problems

The problem can be generalized to operations other than integer addition. Impagliazzo and Naor considered the hardness of subset sum problem in finite commutative groups [63, 64].

If $\psi : X \to G$ is a surjective group homomorphism, then an algorithm that solves the subset G-sum function can be used to invert ψ.

Theorem 3.1 (Impagliazzo and Naor)**.** *A single call to an algorithm $\mathcal{A}$ that solves G-sum problems for fixed $(G,+)$ and k with random a_i can be used to invert any surjective group homomorphism $\psi : X \to G$ on a single point with probability at least $1/k$.*

Proof. Let $y \in G$ be the point where we want to invert ψ. Construct a G-sum instance as follows. First select a random index $1 \le i \le k$ and let $a_i = y$. To define the remaining $k-1$ weights select random $x_j \in X$ for $j \neq i$ and compute $a_j = \psi(x_j)$. Select a random $z \in X$ and obtain a solution $m_1, \dots, m_k$ to the G-subset sum instance

$$\sum_{i=1}^{k} m_i a_i = \psi(z) \ .$$

With probability at least $1/k$ the solution returned satisfies $m_i = 1$. If this is the case, compute

$$\psi^{-1}(y) = z + \sum_{\substack{1 \le j \le k \\ j \neq i}} (-m_j x_j).$$

□

It follows that one expects to need k solutions to random G-sum instances to invert ψ on a single point.

3.4.2 Algorithms

The following applies to additive knapsacks, unless specified otherwise. Complexity of subset sum problems depends on *density* of the instance, defined as $k/\log_2 \max_i a_i$. The quantity is related to the amount of solutions one expects. Lagarias and Odlyzko described an efficient reduction to lattice SVP for instances with density below 0.645 [76]. The bound was improved to 0.9408 by Coster et al. [42].

Instances with density close to one are considered hard. Shamir and Schroeppel developed an algorithm that solves such instances in time $O(2^{k/2})$ and memory $O(2^{k/4})$ [128]. A new algorithm for high density instances was proposed recently by Howgrave-Graham and Joux [62] followed by an improvement by Becker et al. that runs in time $\tilde{O}(2^{0.291k})$ [14].

High density modular subset sums with many solutions were considered by Lyubashevsky [84] and Shallue [129]. The central technique applied is the tree algorithm that we discuss in Section 3.6.1.

3.4.3 Related Cryptosystems

Knapsacks were introduced to cryptography by Merkle and Hellman [90]. The powerful attacks by lattice reduction wiped out most systems of this type. An additive modular knapsack cryptosystem by Chor and Rivest resisted attacks for several years [38]. We will discuss it in more detail in Chapter 7.

3.5 Syndrome Decoding

The problem discussed in this Section is a variant of the subset sum considered in the additive group of binary vectors. Let C be a binary code of length k and dimension $k-n$ with a parity check matrix A. The product Ax for a k-bit x is called the *syndrome* of x. The vector x is a codeword if $Ax = 0$. If a codeword x is received with an error as $x' = x + e$, the syndrome is entirely determined by the error vector e. Decoding is the search for an error pattern e with low Hamming weight. The following problem was shown NP-Complete by Berlekamp et al. [20].

Problem 3.6 (Computational Syndrome Decoding (CSD)). *Given an $n \times k$ binary matrix A, an n-bit s and integer $w \leq k$, find $x \in \{0,1\}^k$ such that x has Hamming weight at most w and $Ax = s$.*

In a related problem one asks for a low-weight codeword.

Problem 3.7 (Codeword Finding (CF)). *Given an $n \times k$ binary matrix A and integer $w \leq k$, find a non-zero word $x \in \{0,1\}^k$ such that x has Hamming weight at most w and $Ax = 0$.*

3.5.1 Algorithms

For w large enough, both the above problems are easily solvable by linear algebra. The fastest known algorithms for solving the hard cases are collectively called Information Set Decoding and follow an idea by Stern [136]. An algorithm due to Canteaut and Chabaud [33] was reported to be the fastest by Overbeck and Sendrier [104]. Efficient decoding algorithms were recently evaluated by Finiasz and Sendrier [48].

3.5.2 Related Cryptosystems

The first public-key encryption scheme based on linear codes was introduced by McEliece [88]. An equivalent construction was described by Niederreiter [103]. Both schemes rely on the hardness of the CSD problem. Neither is known to be widely deployed.

3.6 Generalized Birthday Problem

The problem discussed in this section arises naturally in several areas of computing. It was studied several times along with variants of the elegant solving algorithm. Here we follow the popular exposition by Wagner [147, 146] and recent updates by Minder and Sinclair [95, 96]. We consider the following problem:

Problem 3.8 (Generalized Birthday)**.** *Given a finite group $(G, +)$, an element $y \in G$ and disjoint lists of group elements $L_1, \ldots, L_k$ find $a_i \in L_i$ such that*

$$a_1 + a_2 + \ldots + a_k = y \ .$$

Assume that the elements in the lists L_i were selected at random. Without loss of generality we can work with $y = 0$. An instance with general y can be transformed by adding $-y$ to all the elements in one of the lists. Let $|G| \approx 2^n$ and assume that all the lists are equal in length such that $\log_2 |L_i| = b$. Depending on the context, we will refer to the problem as k-sum or k-product. For appropriate G and L_i the k-sum problem includes several of the specific problems considered in this chapter so far. For example, any subset-sum instance can be seen as a k-sum problem on lists with two elements.

3.6.1 Algorithms

The modern algorithms for solving k-sum problems perform join operations on pairs of lists. The approach was used by Schnorr and Vaudenay to analyze a more general class of related problems [127]. Earlier applications of similar algorithms are discussed in Chapter 4.

We will follow Wagner in developing an algorithm for the problem in several steps. First consider the simple case $k = 2$. Two lists of group elements are given and we are asked to find $a_1 \in L_1$ and $a_2 \in L_2$ such that $a_1 + a_2 = 0$. Assume that the lists are long enough to achieve $|L_1||L_2| > |G|$ or equivalently, $2b > n$. A naive approach would be to start computing all the 2^{2b} sums $a_1 + a_2$ for $a_1 \in L_1$, $a_2 \in L_2$ and check whether the result equals 0. We can do significantly better thanks to the group structure and *meet in the middle*. Check whether any $a_1 \in L_1$ equals $-a_2$ for $a_2 \in L_2$. As the first step, invert all elements in L_2. Then search for a match between L_1 and $-L_2$. This can be achieved by sorting the lists first, faster methods using hash tables are available. This algorithm finds a solution a_1, a_2 in $O(2^b)$ steps.

The above approach looks for pairs a_1, a_2 such that $a_1 = -a_2$. This is equivalent to $a_1 = -a_2 \mod K_0$ or $a_1 + a_2 \in K_0$ where K_0 is the trivial subgroup of G. This is done by checking whether $\rho(a_1) = \rho(-a_2)$ where ρ is the canonical homomorphism $G \to G/K_0$. Generalize this by letting K be any invariant subgroup of G with the appropriate homomorphism $\rho : G \to G/K$. Define an operator $\bowtie_K$ on two lists L_i,L_j that returns the list L_{ij} of all $a_i + a_j$ for $a_i \in L_i$ and $a_j \in L_j$ such that $a_i + a_j \in K$. The operator applies ρ to all $a_i \in L_i$ and computes $\rho(-a_j)$ for all $a_j \in L_j$. Then it returns $a_i + a_j$ such that $\rho(a_i) = \rho(-a_j)$. If L_i and L_j contain random elements of G, the expected length of L_{ij} is $|L_i| \times |L_j| \times |K|/|G|$.

4-list Algorithm

We proceed to describe a tree algorithm for $k = 4$. Assume that $b = n/3$, i.e. the input lists contain $2^{n/3}$ elements each. Suppose there is an invariant subgroup K such that

$$K_0 \subseteq K \subseteq G \ ,$$

$|K| \approx 2^{2n/3}$ and $|G|/|K| \approx 2^{n/3}$. Compute $L_1 \bowtie_K L_2$ and $L_3 \bowtie_K L_4$ to form L_{12} and L_{34}. Each of the two new lists is expected to contain $2^{2n/3} \times |K|/|G| \approx 2^{n/3}$ elements. Then compute $L_{12} \bowtie_{K_0} L_{34}$ to form a single list L_{1234}. The expected length of this last list is $2^{2n/3} \times |K_0|/|K| = 1$. The algorithm performs three merging steps on lists of length $2^{n/3}$ that need at

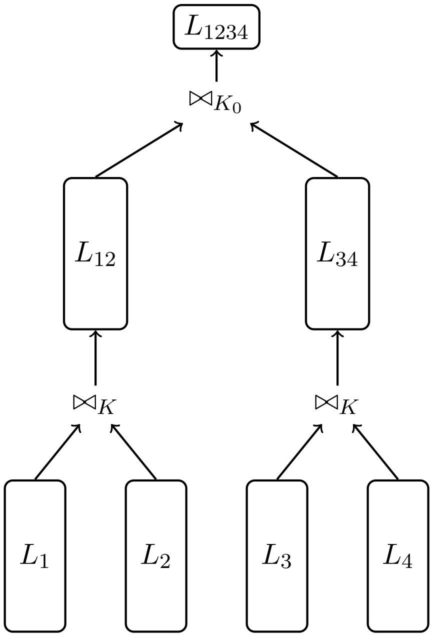

Figure 3.1: The 4-tree Algorithm

most $3 \times 2^{n/3}$ group operations in total. The process is illustrated in Figure 3.1.

k-tree Algorithm

To generalize the above approach further, let $k \geq 4$ be a power of two. Assume that every input list contains 2^b elements for $b = n/(1 + \log_2 k)$. Let there be a chain of normal subgroups

$$\{0\} = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_{1+\log_2 k} = G$$

with the corresponding homomorphisms $\rho_j : G \to G/K_j$. Assume the groups satisfy $|K_j| \approx 2^{bj}$ for $1 \leq j \leq 1 + \log_2 k$. The algorithm proceeds in rounds indexed by j starting with $j = \log_2 k$. Every value of j corresponds to a level in the binary tree sketched in Figure 3.2. In the beginning of a round, there are 2^j lists of elements of the group K_{j+1}. Every list contains approximately 2^b elements. Combine pairs of adjacent lists using the operator $\bowtie_{K_j}$ to obtain 2^{j-1} new lists of elements all confined to K_j. The expected lengths are $2^{2b} \times |K_{j-1}|/|K_j| = 2^b$. Decrement j and repeat the process until j

reaches 0. At that moment, there is a single list of 2^b elements all in K_1. Because $|K_1|/|K_0| \approx 2^b$, expect one of the elements to match 0 modulo K_0. The algorithm runs in $O\left(k \times 2^{\frac{n}{1+\log_2 k}}\right)$ time and space. The number of group operations performed is at most $k \times 2^{\frac{n}{1+\log_2 k}}$.

Runtime for Large k

The length of lists involved in the tree algorithm decreases with growing k. For given $|G|$, the total cost is minimized for $k = 2^{\sqrt{n}}$. The total runtime $O\left(2^{2\sqrt{n}}\right)$ is subexponential in $n = \log_2 |G|$.

Extended k-tree Algorithm

Wagner's analysis assumes that the lists are long enough, i.e. $b(1+\log_2 k) \geq n$. The tree algorithm is likely to fail otherwise. Minder and Sinclair analyzed instances with shorter initial lists under the condition that a solution is expected to exist [95, 96]. Generalize the problem to ask for 2^c solutions instead of a single one for $c < 2b$. To expect this many solutions, require that $bk \geq n + c$. In order to solve the k-sum problem for instances where $b(1 + \log_2 k) \leq n + c$, keep all the sums $a_i + a_j$ on the t bottom levels of the tree. This causes the lists to grow rapidly. Each list contains approximately 2^{b2^t} elements after t such rounds. In the remaining $\log_2 k - t$ rounds apply Wagner's original algorithm that filters partial sums modulo subgroups of G. Minder and Sinclair prove that the runtime of such a modified algorithm is optimal if t is the least integer that satisfies

$$n + c \leq (\log_2 k - t + 1)b2^t \tag{3.2}$$

and the lists are allowed to grow up to the maximum length of 2^u elements where

$$u = \frac{n + c - b2^t}{\log_2 k - t} \ . \tag{3.3}$$

As before, the complexity is proportional to the number of lists k multiplied by the maximum list length 2^u. The extended k-tree algorithm runs in time $O(k \times 2^u)$.

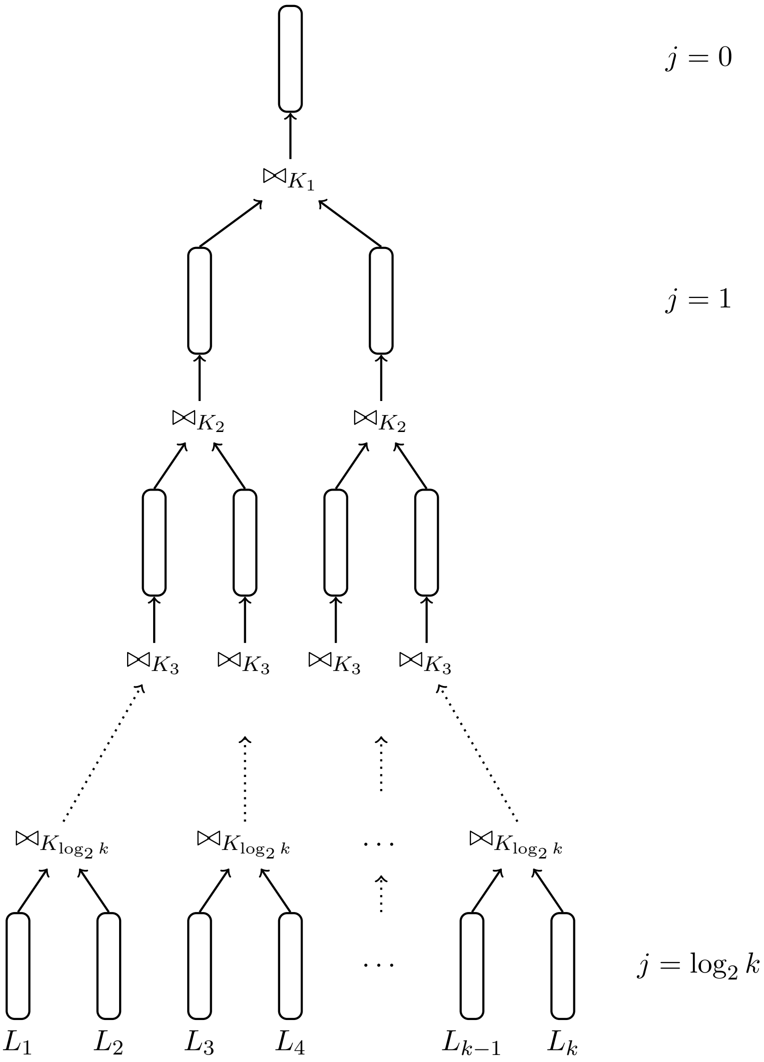

Figure 3.2: The k-tree Algorithm

Structure of G

The k-sum algorithms for $k \geq 2$ depend on the availability of the subgroups K_j and the corresponding homomorphisms ρ_j. The group $(\{0,1\}^n, \oplus)$ originally considered by Wagner is an example where many subgroups are available. There is a subgroup of order 2^l for all integers $0 \leq l \leq n$. On the other hand, there are groups that have no non-trivial subgroups. An example of such a group $\mathbb{Z}_p$ for p prime. Wagner shows that the tree algorithm can be adapted to run in this group if the subgroups K_j are replaced by the chain of intervals $U_j = [-u_j, u_j - 1]$ such that $|U_j| \approx |G|^{\frac{j}{1+\log_2 k}}$ and $U_j \subseteq U_{j+1}$. The join operators $\bowtie_{U_j}$ compute the pairs $a_1 + a_2$ such that $a_1 + a_2 \in U_j$. The same construction can be extended to any additive Abelian group represented as $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \times \mathbb{Z}_{n_l}$.

Wagner relaxed the requirement on G itself to be a group and considers the k-sum problem for several non-group operations. Such generalizations appear not to be relevant for this thesis.

The Case of XOR

The k-sum problem in an additive group of n-bit binary strings can easily be solved by linear algebra if k is large compared to n. In order to do so, select two elements α_i and β_i per list, ignore all remaining elements in L_i. Compute the n-bit vectors $\gamma_i = \alpha_i + \beta_i$ and $y = \sum_{i=1}^{k} \beta_i$. Solve the linear system $v_1\gamma_1 + \ldots + v_k\gamma_k = y$ over the binary field. A solution is likely to exist if $n \leq k$. The elements $x_i = \beta_i + v_i\gamma_i$ then solve the corresponding k-sum problem.

Wagner suggests to use linear algebra even in cases $k \leq n$ in a hybrid approach by transforming the instance to a smaller group. An instance on k lists of n-bit strings can be transformed to an instance on k' lists of $n-k+k'$ bit strings for $0 < k' < k$. Such a transformation appears not to be worth the effort for k significantly smaller than n.

3.6.2 Lower Bounds

In contrast to the problems in the previous sections, the k-sum problem itself has not served as an assumption supporting the security of any public-key scheme. It is nevertheless possible to connect it to many of the "standard" hard problems.

Wagner points out that hardness of the DL problem in G implies that the k-product is hard in a Theorem attributed to Wei Dai.

Theorem 3.2. *If the k-sum problem over a cyclic group $G = \langle g \rangle$ can be solved in time t, then the discrete logarithm with respect to g can be found in $O(t)$ time as well.*

Proof. Let $y \in G$ be arbitrary. We will describe how to find $\log_g y$ given an algorithm that solves k-products in G. Fix k and generate lists L_i that contain g^u for random exponents u. A solution to the k-product instance of the form $\prod_{i=1}^{k} x_i = y$ corresponds to an additive relation $\sum_{i=1}^{k} \log_g x_i = \log_g y$. Compute $\log_g y$ from the logs on the left hand side that are known by construction. □

The lower bound on the cost of discrete logarithms implies that any algorithm that solves the k-sum problem generically runs in $\Omega(\sqrt{p})$ for p the greatest divisor of $|G|$.

Theorem 3.1 can be extended to the k-sum setting. A solution to the k-sum problem in G is by definition a solution to the corresponding G-sum. The complexity of k-sum problem in $\mathbb{Z}_M^*$ is connected to the hardness of factoring M.

An efficient algorithm for solving k-sum problems in a broader spectrum of groups would likely be able to solve all the hard problems considered in this chapter so far.

Upper Bounds

According to Wagner, the connection to discrete logarithms goes both ways. If G is a cyclic group where the DL problem is easy, the tree algorithm can solve k-product instances over G as fast as in the corresponding additive group $(\mathbb{Z}_M, +)$ for $M = |G|$.

For well chosen parameters this can still involve a considerable effort. For most groups there appear to be no known algorithms that would solve k-sum problems faster than the tree algorithm. The case of the XOR operation with k big relative to n is a clear warning, but as far as available attacks are concerned, the general k-sum problem looks promising.

Chapter 4

Early Hash Functions in Groups

A group is probably one of the simplest and most abundant of all mathematical structures. Integer addition or multiplication are group operations on appropriately defined sets. The two very natural operations on numbers to perform are supported by virtually every computer. So is the binary XOR operation, i.e. component-wise vector addition modulo 2. A look at "ordinary" hash functions suggests that *all* of them are built on finite group operations in one way or another. What makes the functions considered in this thesis special? It is the simplicity, the fact that they are based almost exclusively on a single group operation. Their security properties are equivalent to problems that are very simple to formulate and possibly also easy to comprehend.

This chapter surveys early proposals of hash functions in finite groups, in particular those that preceded cryptanalysis of MD5 and SHA by Wang et al. [149]. The concept of group knapsacks was reinvented several times in the past few decades. We emphasize the structure and common features over details on implementations or speed. After all, none of the functions has survived to become used in practice and most of them are broken. This chapter is intended to provide a partial historical account of the developments to set our new compression functions into proper context.

The notation used in this chapter may differ considerably from the notation used in the original proposals we refer to. Several functions were deliberately rewritten in order to improve consistency of presentation within the thesis and to stress the features present across several functions.

4.1 Matrix Multiplication by Bosset

Probably the first proposal to build a cryptographic hash function in a finite group was that of Bosset [27]. The scheme was actually called a *signature* rather than a hash function.

Let $(G, *)$ be a non-Abelian group, define a fixed set $A = \{a_1, \ldots, a_k\} \subseteq G$. Select the a_i at random under the condition that $a_i * a_j \neq a_j * a_i$ for $i \neq j$. This condition also implies that no a_j is an inverse of a_i for $i \neq j$. For simplicity let $k = 2^b$ and let $\phi : \{0,1\}^b \to A$ be a fixed injective function. Assume further that the length of the input message m is divisible by b, denote the i-th b-bit block of m by $m[i]$. To hash an l-bit message proceed as follows:

1. Initialize $x_1 = 1_G$.
2. Let $\mathcal{L} = \frac{l}{b}$.
3. For $i = 1, \ldots, \mathcal{L}$ compute $x_{i+1} = x_i * \phi(m[i])$.
4. Return $x_{\mathcal{L}+1}$.

There is no limit on the length and l does not even need to be known in advance. The digest can be computed "on-the-fly" in a single pass through the input data. The mapping ϕ only encodes the message in a new alphabet A. The hash computation then amounts to a $*$-product of message characters in $(G, *)$. Note that for messages long enough some of the a_i will repeat potentially many times.

Choice of G In a concrete instance, Bosset selected the multiplicative group $GL(2, \mathbb{F}_p)$ of 2×2 non-singular matrices over a finite field $\mathbb{F}_p$ for p prime. Concrete parameters proposed were $p = 997$, $k = 64$ and $b = 6$. Size of the output is approximately $4 \log_2 p \approx 40$ bits. Every 6-bit string is assigned an element of A, one needs a single matrix multiplication per 6 bits of input.

The following three simple consequences of the group structure are independent of the choice of G:

Associativity For any triple of messages m_1, m_2, m_3 it holds that

$$H(m_1||m_2) * H(m_3) = H(m_1||m_2||m_3) = H(m_1) * H(m_2||m_3) \ .$$

Incrementality Given two messages m_1 and m_2, the digest of their concatenation $m_1||m_2$ can be computed from $H(m_1)$ and $H(m_2)$ as

$$H(m_1||m_2) = H(m_1) * H(m_2) \ .$$

If a message is extended, we only need to hash the "fresh" bits and multiply with the original digest. In an analogy, if a message is truncated, we can divide the digest by the hash of the removed part.

Non-Commutativity If the messages m_1 and m_2 are not identical, then $H(m_1||m_2)$ in general differs from $H(m_2||m_1)$. By a simple counting argument there are plenty of exceptions. The odds of finding one at random should be minimal. Note that if the group G was commutative, collisions in H would be trivial to find.

Security properties

The security properties considered were first and second preimage resistance. Cost of exhaustive search was measured and the function was concluded secure against the generic attacks. Collision resistance was not considered. The generic collision search algorithm would have needed hashing about a million messages, likely to be within the limits of computing power available.

Given $y \in G$, preimage search is equivalent to finding an factorization of y in terms of elements of A. Pairs of colliding messages correspond to non-trivial factorizations of the unit in G. Several such factorizations are easy to describe, for example $a_i^{|G|} = 1_G$ for all i. Such messages are very long and should not be considered a threat to security. After all, it is much faster to complete a birthday collision search than it is to print a pair of colliding messages that are $|G|b$ bits long combined.

Separation of Colliding Inputs If an input message is modified such that a single a_i is replaced by $a_j \neq a_i$ then the digest necessarily changes. Such a change happens when precisely one of the b-bit message blocks is replaced. If two different consecutive b-bit blocks are swapped, the digest changes as well. Minor changes of the above types *always* change the resulting hash value. The property follows from group cancellation laws.

4.1.1 Cryptanalysis

The function was broken by Paul Camion in a preimage attack that was probably the earliest instance of the tree algorithm described in Section 3.6.1 [30]. Camion kept $k = 64$ and raised p to 10007 to make the problem harder. The chain of subgroups

$$\{I\} = K_0 \subseteq K_1 \subseteq K_2 \subseteq K_3 \subseteq K_4 = G$$

needed by the tree algorithm is defined as follows:

- K_1 is the group of matrices $\begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix}$ for $b \in \mathbb{F}_p$.
- K_2 is the group of matrices $\begin{pmatrix} a & 0 \\ b & 1 \end{pmatrix}$ for $a, b \in \mathbb{F}_p$.
- K_3 is the group of matrices $\begin{pmatrix} a & 0 \\ b & d \end{pmatrix}$ for $a, b, d \in \mathbb{F}_p$ and $ad \neq 0$.

Observe that $K_{i-1} \subseteq K_i$ and $|K_i|/|K_{i-1}| \approx \sqrt[4]{|G|}$, the logarithms of $|K_i|$ are equally spaced between 0 and $\log |G|$.

The attack starts with 16 lists of approximately 12000 elements each and performs four merging steps to find a single solution to the 16-sum problem with an arbitrary target $y \in G$. Elements of the lists are constructed from products of random triples of generators from A. There are 64^3 candidate list elements in correspondence with 18-bit input blocks. A solution to the 16-sum problem then leads to an 288-bit input message.

4.2 Two Schemes by Godlewski and Camion

4.2.1 Integer Addition

An additive knapsack compression function was considered by Godlewski and Camion [55]. Fix an integer k and a bound M. Select k random integral weights $a_i < M$. To hash a message m composed of k input bits m_1 to m_k compute

$$H(m) = \sum_{i=1}^{k} m_i a_i.$$

In order for the function to compress it is necessary that $\log_2 M + \log_2 k < k$. Godlewski and Camion proceed to shown the scheme is insecure if $4 \log_2 M < (\log_2 k)^2$, i.e. when the density of the knapsack is too high. A deterministic algorithm finds second preimages in about $4k \log_2 k$ additions of M-bit integers.

4.2.2 Error Correcting Codes

Another related construction is based on an error correcting code over a finite field $\mathbb{F}$. Let C be a $[k + t, k, d]$ code defined by the generating matrix $E =$

$\big(I_k|A\big)$ such that I_k is the $k \times k$ identity matrix over $\mathbb{F}$ and the size of A is $k \times t$. Fix a collection of injective one way functions $\phi_i : \{0,1\}^b \to \mathbb{F}$ for $i = 1$ to k. To compute the digest of a bk-bit message m, proceed as follows:

1. Compute $\gamma_i = \phi_i(m[i])$ for $i = 1$ to k and let $\gamma = (\gamma_1, \ldots, \gamma_k)$.
2. Encode the vector γ under C as $\gamma E = (\gamma_1, \ldots, \gamma_k, \gamma'_{k+1}, \ldots, \gamma'_{k+t})$.
3. Output the t redundancy symbols $\gamma'_{k+1}, \ldots, \gamma'_{k+t}$.

The resulting t-dimensional vector is equal to the matrix product γA. Given an $y \in F^t$, a preimage under A can be found easily by linear algebra. For this reason the one-way injections ϕ_i were used.

Separation of Colliding Inputs By the properties of the code, any two codewords must differ in at least d entries. If two non-identical codewords collide on the last t positions, they necessary differ in at least d from among the first k positions. If two different messages have identical digests, the message vectors over $\mathbb{F}$ differ in at least d entries. By injectivity of ϕ_i, the messages need to differ in at least d corresponding b-bit blocks. The authors consider MDS codes that achieve $d = t + 1$. It is suggested that $|F| > 2^{100}$, $k + t < 2^{32}$ and $2 \leq t \leq 10$. This allows potentially more separation of colliding inputs than in the case of Bosset's function.

4.2.3 Cryptanalysis

We can rewrite the function in the following more familiar form:

1. Initialize $x_1 = 0 \in F^t$.
2. For $i = 1$ to k compute $x_{i+1} = x_i + \gamma_i A_i$.
3. Output x_{k+1}.

The expression A_i stands for the i-th row of A and the addition is performed in the additive group $(F^t, +)$. If we define $L_i = \big\{\phi_i(x)A_i | x \in \{0,1\}^b\big\}$, then the inversion of H is an instance of the k-sum problem. If the characteristic of F is small, the problem can be solved efficiently by linear algebra. For high characteristic, Godlewski and Camion adapt the tree algorithm from [30]. If the length of blocks is big enough, the function can be inverted in about $2^{2\sqrt{n}}$ steps by varying approximately $2^{\sqrt{n}}$ input symbols for $n \approx \log_2 |G|$ (cf. Section 3.6.1).

4.3 Matrix Multiplication by Tillich and Zémor

Zémor modified the scheme of Bosset by fixing $b = 1$ and $k = 2$ and working in a subgroup of $GL(2, \mathbb{F}_p)$ [153, 152]. Let $G = SL(2, \mathbb{F}_p) \subseteq GL(2, \mathbb{F}_p)$ be the multiplicative group of 2×2 matrices over F_p with determinant 1. The set A of generators only contains the two elements

$$a_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \; a_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

such that $\phi(0) = a_1$ and $\phi(1) = a_2$. The algorithm is otherwise identical to that of Bosset. The specific matrices allow faster hashing than a random choice would. No field multiplications are needed, one only needs to perform additions of integers modulo p. The group G has approximately p^3 elements. The suggested length of p is 150 bits, the output would be $3 \log_2 p \approx 450$ bits long.

Cayley Graph Interpretation Consider the directed 2-regular Cayley graph associated to G and the generator set A. The hash function iteration can be seen as a walk starting in the vertex representing the identity in G. For every input bit m_i follow the edge labeled $\phi(m_i)$. Output the group element represented by the final vertex reached.

Separation of Colliding Inputs

The separation property derived by Bosset is preserved. A bit flip as well as a swap of two adjacent bits always change the result. The special choice of A allows to prove more.

Proposition 4.1. *If $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$ are two different bit strings colliding under H, then $\max(i, j) \geq \delta$ such that $\delta = \log_\alpha \frac{p}{2}$ for $\alpha = \frac{1+\sqrt{5}}{2}$.*

Proof Sketch. Omit the modular reduction and compute the products

$$\begin{aligned} X &= \phi(x_1)\phi(x_2)\ldots\phi(x_i) \\ Y &= \phi(y_1)\phi(y_2)\ldots\phi(y_j) \end{aligned}$$

over the ring of integers and show that $X \neq Y$ when unreduced. On the other hand, $X - Y = 0$ modulo p. Therefore at least one entry in $X - Y$ is an integral multiple of p. Taking norms it follows that $||X - Y|| \geq p$ and

$$\max(||X||, ||Y||) \geq \frac{p}{2} \; .$$

The norm of a_1 and a_2 is equal to α. The result follows from sub-multiplicativity of matrix norms. □

Corollary 4.2. *If a message m' is obtained from m by replacing i consecutive bits by a different string of j bits for i, j as in Proposition 4.1, then $H(m) \neq H(m')$.*

For a 150-bit p, the value of δ is slightly less than 215. In the case of Bosset's function, colliding inputs had to differ in at least two 6-bit blocks. With the function by Zémor and Tillich, one is guaranteed to get distinct results when hashing all the 2^{215} messages that are at most 214 bits long.

Security

The function was designed to offer collision resistance, but the original choice of A turned out to be insecure. Tillich and Zémor devised a fast collision finding algorithm [138]. Considering the computation over $\mathbb{Z}$ a short factorization of the identity in terms of a_1 and a_2 can be found with the help of Euclid's algorithm. Such factorizations always lead to colliding messages. The authors proposed to replace the generators in A by their small integral powers to prevent the attack.

Camion's attack applying on Bosset's function was claimed to be no threat, because the special form of generators allowed to choose big p and consequently large $|G|$. The change in G itself would also require a new chain of subgroups K_i.

Binary Fields

The field $\mathbb{F}_p$ was replaced by a binary field in [139]. Let $p(t)$ be an irreducible polynomial of degree $d > 1$ over $\mathbb{F}_2$ and define $\mathbb{F} = \mathbb{F}_2[t]/(p(t))$. The two generators in A were replaced by the pair

$$a_1 = \begin{pmatrix} t & 1 \\ 1 & 0 \end{pmatrix}, \; a_2 = \begin{pmatrix} t & t+1 \\ 1 & 1 \end{pmatrix} .$$

Proposition 4.1 adapted to the new setting leads to $\delta = d + 1$. To obtain the result, one looks at the maximal degree of polynomials within the matrices. The generators in A have degree one, Tillich and Zémor show that in order to obtain equality mod $p(t)$ one needs to reach at least degree d.

The suggested values for the degree d were $130 - 170$, the corresponding output sizes would be 390-510 bits.

4.3.1 Cryptanalysis

The proposal did not restrict $p(t)$ in any way. Charnes and Pieprzyk exploited this and showed how to construct $p(t)$ such that one of the generators in A has low degree [36]. Collisions are trivial to construct for such an instance.

Steinwandt et al. exhibited more choices of $p(t)$ that were vulnerable [135]. Collisions can easily be computed if a non-trivial decomposition $p = g(h(t))$ is known. This requires d to be composite, the attack can be avoided if d is prime.

In more recent paper, Grassl et al. described an efficient collision-finding algorithm that is independent of the choice $p(t)$ [59]. They lift the computation to the ring of matrices over the polynomial ring $\mathbb{F}_2[t]$ and find colliding messages only $2d + 2$ bits long. The attack was generalized by Petit and Quisquater to find preimages efficiently [108].

4.4 Group Subset Sums by Impagliazzo and Naor

Families of compression functions in general finite groups were also considered by Impagliazzo and Naor [63, 64].

Definition Let $(G, +)$ be a finite group. Let k be an integer such that $|G| < 2^k$. Select k distinct elements of G at random, denote them by $a_1, \ldots, a_k$. Let m be a k-bit input message consisting of bits m_1 to m_k. Define a compression function $H : \{0, 1\}^k \to G$ as follows:

$$H(m) = \sum_{i=1}^{k} m_i a_i \ , \tag{4.1}$$

where the sum is computed in $(G, +)$.

If we substitute an additive group of integers for G, the function computes “standard” subset sum and any hardness assumptions on such problems transfer to statements on preimage resistance of H.

For some choices of G the authors proceed to prove security based on number-theoretic problems using Theorem 3.1.

Integer Multiplication

Given a group $G = \langle g \rangle$ the mapping $i \mapsto g^i$ is an onto homomorphism. Inverting a subset sum function G is therefore at least as hard as a fraction $1/k$ of the cost of computing discrete logarithms.

A preimage resistant function can therefore be built in any group where logarithms are hard to compute, such as multiplicative groups of finite field described in Section 3.2.

The function $x \mapsto x^2$ in the group of quadratic residues modulo M is an onto homomorphism. Square roots modulo M are known to be as hard as factoring. A subset product on k random quadratic residues mod M is at least as hard as $1/k$ of the cost of factoring M.

Impagliazzo and Naor also showed that the subset product function modulo $M = pq$ for p, q prime can be reduced to subset product in the group of quadratic residues modulo M. One can therefore pick the a_i from all of $\mathbb{Z}_M$, the choice is not limited to the quadratic residue subgroup. Removal of this restriction introduces at most a k-fold drop in security.

No explicit parameter values were proposed. An instance of the above function would need to store k group elements and performs a single group operation per bit of input.

4.5 Integer Addition by Damgård

In order to avoid the attack on the additive function from Section 4.2.1, Damgård proposed to limit the compression ratio and therefore the knapsack density [44]. The compression function would be turned to a full hash function by the means of the new mode of operation (cf. Section 2.5). Concrete parameters proposed for the function were $k = 256$ and $M = 2^{120} - 1$. The sum

$$H(m) = \sum_{i=1}^{k} m_i a_i$$

is then at most 128 bits long. Total size of the weights is under 4 kB, the function performs a single 128-bit addition per input bit. With a 256-bit input the function compresses by a factor of two.

4.5.1 Cryptanalysis

No explicit bounds on security of the compression function were proved by the designer. Camion and Patarin described an inversion algorithm that is an 8-list instance of the tree algorithm from Section 3.6.1 [31]. Select a modulus $M' \approx 2^{128}$ such that $M' = \prod_{i=1}^{4} M_i$ for $M_i \approx 2^{32}$ that are pairwise coprime. The modulus M' is greater than the maximum possible value of an output of H, we can consider the function to operate in the additive group $\mathbb{Z}_{M'}$.

By the Chinese Remainder Theorem there is an efficient isomorphism to the additive group

$$\mathbb{Z}_{M_1} \times \mathbb{Z}_{M_2} \times \mathbb{Z}_{M_3} \times \mathbb{Z}_{M_4} \ .$$

This representation provides the chain of subgroups needed to run the tree algorithm. The attack starts with 8 lists of 2^{32} integers. Split the weights a_i in eight subsets of 32 and fill each list with all the sums over a subset of weights. Three merging steps result in a single list of $2^{32} \approx M_4$ elements all equal to the target sum y modulo $M_1 M_2 M_3$. One solution can be expected to match modulo M_4 as well. This solves the subset sum instance and leads to a bit string that hashes to y.

The lists would occupy approximately 64 GB of space. The attack algorithm was not implemented.

Lattice Attacks The underlying subset sum instance was beyond the reach of lattice reduction algorithms available at the time it was proposed. When improved algorithms appeared [125], Joux et al. described an approach to find collisions in the full hash function [66, 69]. To allow any value in the chaining variable, only the subset-sum instance formed by the last 128 weights was attacked. The density of the instance is approximately 1.

A later reduction algorithm led to even more efficient collision search attacking only 128 weights [126].

4.6 Vector Addition by Goldreich et al.

Security of the family of compression functions described in this section is related to hardness of lattice problems and follows Ajtai's construction sketched in Section 3.3.3. Let q be a prime number, and k, d be integers such that $k > d \log_2 q$, $k < \frac{q}{2d^4}$ and $q = O(d^c)$ for a constant c. Select a random $d \times k$ matrix A with entries from $\mathbb{Z}_q$. Let m be a k-bit input message, here interpreted as a k-dimensional vector over $\mathbb{Z}_q$. Define a compression function as

$$H(m) = Am = \sum_{i=1}^{k} m_i a_i,$$

where a_i is the i-th column of A and all the computations are performed modulo q. The output is a d-dimensional vector over $\mathbb{Z}_q$. This function matches the general definition (4.1) where G is the additive group $(\mathbb{Z}_q)^d$.

Note that the function is linear, therefore given $y \in (Z_q)^d$ it is trivial to find $x \in (\mathbb{Z}_q)^k$ such that $Ax = y$. Only the solutions with entries limited to

$0, 1$ are preimages of y under H. A non-zero x with entries $\{-1, 0, 1\}$ such that $Ax = 0$ corresponds to a collision in H. Construct messages m, m' such that $m_i = 1$ if $x_i = 1$ and $m'_i = 1$ if $x_i = -1$ and $m_i = m'_i$ are identical on bit positions where $x_i = 0$.

By Ajtai's result (Section 3.3.3), the function is hard to invert on average if SVP is hard to approximate in the worst case. Goldreich et al. pointed out that H is collision resistant as well [56]. No explicit parameter choices or concrete security bounds were provided.

4.7 Fast Syndrome Based Hash

In this section we define the early variant of the Fast Syndrome Based compression function proposed by Augot et al. [8]. Let $G = ((\mathbb{Z}_2)^n, \oplus)$ be the group of all n-bit binary strings with the XOR operation. Let b, k be positive integers such that $bk > n$. Select a random binary $n \times k2^b$ matrix A. Define functions $\phi_i : \{0, 1\}^b \to \{0, 1\}^n$ for $i = 1$ to k that interpret the input as a binary expansion of an integer and $\phi_i(x)$ returns the column number $(i - 1)2^b + x + 1$ from the matrix A.

If the i-th b-bit chunk of m is denoted by $m[i]$, the FSB Compression function can be defined in the following familiar form:

1. Initialize $x_1 = 0 \in \{0, 1\}^n$.
2. For $i = 1$ to k compute $x_{i+1} = x_i \oplus \phi_i(m[i])$.
3. Output x_{k+1}.

The function involves $k-1$ XOR operations on n-bit strings, the total size of the defining matrix is $nk2^b$ bits. If we first transform the input message to a $k2^b$-bit string s that has value 1 precisely on the bit positions numbered $(i-1)2^b + m[i] + 1$ for $i = 1$ to k, the function can be seen as a simple matrix multiplication

$$H(m) = As \ . \tag{4.2}$$

This function computes the hash value as a sum of k group elements, but the number of "weights" in the knapsack is increased. The function needs only a single group operation per b message bits, but it stores $k2^b$ group elements.

4.7.1 Security

Augot et al. proved the function secure under an assumption on the hardness of syndrome decoding problems. We shall discuss the argument in more

Table 4.1: FSB and the Tree Algorithm

b	k	n	list length	# of lists	cost
8	64	160	2^{30}	32	2^{36}
8	96	224	2^{38}	32	2^{43}
6	128	288	2^{58}	16	2^{62}

detail in Chapter 5.

The matrix product itself (4.2) is trivial to invert. However, only certain $k2^b$-bit vectors correspond to input messages. In order for such a vector to be valid, its weight must be equal to k and precisely one bit has to be set in each of the k consecutive 2^b-bit windows. Such words are called *regular*. Augot et al. considered the complexity of an Information Set Decoding algorithm to derive secure values of the parameters b, k, n. The three parameter sets proposed are captured in the first three columns of Table 4.1. Attacks were claimed to cost more than 2^{80} binary operations. In case of the first function this applies to preimage search. For the two other proposals, it holds for collision search as well.

4.7.2 Cryptanalysis

Coron and Joux applied the tree algorithm to find collisions in the three FSB variants [41]. Originally there are k lists L_i of 2^b elements each. Size of the target group is 2^n. For each list L_i form a new list L_i' that contains sums of precisely two distinct elements from L_i. Every solution to the k-sum instance over L_i' with the target 0 corresponds to a pair of colliding messages.

Length of L_i' is approximately 2^{2b-1}. The tree algorithm in this setting has little chances of success, because $2b - 1 < n/(1 + \log_2 k)$ for all three variants. Coron and Joux therefore first combine the lists L_i' to reduce their number and increase the length. This attack is an instance of the extended tree algorithm described in Section 3.6.1. The number of lists, their length and the total claimed complexity of the attack are captured in Table 4.1. Note that the last attack modifies at most 576 bits out of the 768 bits of input. The remaining 192 bits of the colliding messages must be equal, but can be chosen freely.

4.8 Incremental Functions by Bellare and Micciancio

Bellare and Micciancio proposed a family of incremental hash functions in finite groups [17]. Contrary to other proposals in this chapter, a secure compression function is not only the goal of the construction, but also a point of departure. The goal is to provide a mode of operation that allows fast and simple updates of hash values.

Let $(G, *)$ be a finite commutative group. Fix a block size b and assume $\varphi : \mathbb{N} \times \{0,1\}^b \to G$ is a collision resistant compression function. To hash a bk-bit message m proceed as follows:

1. Initialize $x_1 = 1_G$.
2. For $i = 1, \ldots, k$ compute $x_{i+1} = x_i * \phi(i, m[i])$.
3. Return x_{k+1}.

The central feature of the construction is incrementality. If a block $m[i]$ in a message m is replaced by $m'[i]$, the digest of the new message can be easily computed from $H(m)$ as

$$H(m) * \phi(i, m[i])^{-1} * \phi(i, m'[i]) \ .$$

If the function ϕ is modeled as a random oracle, the incremental construction results in a collision resistant hash function provided the following problem is hard:

Problem 4.1 (Balance Problem). *Given random elements $a_1, \ldots, a_l$ of a group G find two disjoint sets $I, J \subseteq \{1, \ldots, l\}$ not both empty such that*

$$\prod_{i \in I} a_i = \prod_{j \in J} a_j$$

where the product is computed in G.

Bellare and Micciancio proceed to investigate several choices of the underlying group G and the resulting compression functions.

XHASH

If G is the group $((\mathbb{Z}_2)^n, \oplus)$ of n-bit binary strings with the XOR operation, the function was shown to be insecure as soon as k exceeds n. Collisions can easily be found using linear algebra in that case (cf. Section 3.6.1), therefore this choice of G is not suitable for use in the construction.

MuHASH

If G is a multiplicative group such as $\mathbb{Z}_p^*$ for p a suitable prime, the Balance problem in G is hard if discrete logarithms are hard to compute. The setting is analogous to that in Section 4.4. The authors point out that even if discrete logarithms were easy, the function MuHASH would likely remain secure.

AdHASH

Bellare and Micciancio also considered the additive group of integers modulo M in place of G. Such a function is one-way by the arguments from Section 4.4. The authors need to make a new assumption to prove the function collision free. They relate the hardness of the underlying problem to lattice SVP and argue that even if SVP were easy, the knapsack problem may remain hard.

LtHASH

The group $\big((\mathbb{Z}_p)^d, +\big)$ earlier proposed by Goldreich et al. was also considered. The security argument is similar as in Section 4.6. The compression function of Goldreich et al. restricted the input length and required a single group operation per input bit. The variant by Bellare and Micciancio allows longer messages and can get much faster, because one only needs a single operation per b input bits.

4.8.1 Cryptanalysis

For a message bk bits long the scheme computes a sum of precisely k group elements, every single of them is selected from approximately 2^b possible outputs of ϕ. Inversion of the hash function is an instance of the k-sum problem in G.

Wagner applied the tree algorithm to AdHash to find preimages in approximate time $2^{2\sqrt{\log_2 M}}$ [147, 146]. In order to achieve 2^{80} security, the modulus M would have to be about 1600 bits long.

Note that if discrete logarithms are easy in G, the function MuHASH can easily be transformed to an instance of AdHASH and the same subexponential attack applies.

Chapter 5

Knapsacks Revisited

The cryptanalysis of MD5 and SHA-1 resulted in more intensive hash function research and led to renaissance of several constructions, including knapsack-type hash functions. Most of the proposed functions advertised some sort of *provable* security. We summarize the recent developments and pay special attention to any quantified security levels and their connection to the proofs provided. In doing so, we always assume the implicit cost models used by the designers and directly adopt any complexities computed. Parts of this chapter were published in the article *Interpreting Hash Function Security Proofs* [122].

5.1 New Variants of FSB

Recall the basic FSB from Chapter 4 that hashes bk bits by computing a XOR of precisely k binary vectors of length n. There are a total of 2^b choices for each vector in the sum. As a response to the generalized birthday attack by Coron and Joux described in Section 4.7, Augot et al. reconsidered the parameter choices for FSB and proposed new variants with longer output [9]. The total size of the associated defining matrix is $nk2^b$ bits, this update to FSB therefore leads to increased storage requirements. For reasonable speed and security levels, the matrix A has prohibitive size of several megabytes, in a few cases even hundreds. In a later improvement to FSB, Finiasz et al. replace the random matrix A by a quasi-cyclic matrix with compact description and argue that this change has no negative impact on security [47]. To define such a matrix select the first row at random and compute the remaining rows as cyclic shifts by multiples of a fixed integer constant u. The update to the design reduces the space needed by a factor

Table 5.1: FSB Parameters for SHA-3

	b	k	n
FSB$_{160}$	14	80	640
FSB$_{224}$	14	112	896
FSB$_{256}$	14	128	1024
FSB$_{384}$	13	184	1472
FSB$_{512}$	13	248	1984

of n. Some of the parameter sets satisfied $n \leq k$ and therefore allowed the underlying k-sum problem to be solved fast using linear algebra. In order to find collisions, it is sufficient that $n \leq 2k$. The attack was (re-)discovered and implemented by Saarinen [121]. Fouque and Leurent found a weakness in the quasi-cyclic matrix for n composite [50]. The attacks were followed by another update to the FSB design.

5.1.1 SHA-3 Candidate

A variant of FSB was submitted to the NIST SHA-3 competition by Augot et al. [7]. The proposal was not selected for the second round. It consists of an iterated compression function followed by a final transformation to compress the output further. The operation of the compression function is identical to FSB as it is described in Section 4.7. The parameters are selected with respect to all the attacks we mentioned so far. The $n \times k2^b$ matrix A is generated from $k2^b/n$ vectors of p bits for prime $p > n$. Any column of the matrix can be efficiently computed as a cyclic shift of one of the vectors truncated to n bits. The FSB compression step is followed by a single application of a compression function g to reach the desired output length. The mapping g is instantiated by the hash function Whirlpool [12]. The five parameter sets proposed are listed in Table 5.1. The superscript l in FSB$_l$ denotes the final desired output length that is achieved by compressing the n bits by the function g in the final transformation.

Security Claims

The function is proved preimage resistant reducing to the Computational Syndrome Decoding problem and collision resistant reducing to the Codeword Finding problem. Both proofs are immediate. Preimage search is easily seen to be a special case of CSD for the matrix A and weight k, collision

Table 5.2: Cost of Decoding Problems for FSB

	CSD		CF	
	ISD	Tree	ISD	Tree
FSB_{160}	$2^{211.1}$	$2^{163.6}$	$2^{100.3}$	$2^{119.2}$
FSB_{224}	$2^{292.0}$	$2^{229.0}$	$2^{135.3}$	$2^{166.9}$
FSB_{256}	$2^{330.8}$	$2^{261.6}$	$2^{153.0}$	$2^{190.7}$
FSB_{384}	$2^{476.7}$	$2^{391.5}$	$2^{215.5}$	$2^{281.0}$
FSB_{512}	$2^{687.8}$	$2^{527.4}$	$2^{285.6}$	$2^{378.7}$

search is a special case of CF with $w = 2k$. If the two problems are assumed hard for matrices such as A, this trivially implies the two security properties for f. There are however no explicit lower bounds tied to the assumptions, therefore no lower bounds on security are derived. The designers assessed security further looking at possible attacks only.

A detailed analysis of algorithms solving the underlying CSD and CF instances was performed. One of the algorithms is a variant of Information Set Decoding (ISD), the other is Wagner's tree algorithm. The cost levels are displayed in Table 5.2.

The values above are reported as the best attacks against the compression function of FSB. If they were attacks the quantities would become type **U** bounds in our notation. However, the ISD and Tree algorithm attacks considered only solve the *more general* underlying CSD and CF instances. Any $k2^b$-bit vector x with weight at most k such that $Ax = y$ is good enough for CSD. In order for x to be a preimage under H it also has to be *regular*. That means precisely one bit set in each of the k consecutive windows of 2^b bits. Similarly, pairs of colliding inputs only form a proper subset of solutions to the corresponding CF problem. The quantities in Table 5.2 do not necessarily correspond to cost of actual attacks and only constitute **uL** bounds. Breaking the compression function can still be *harder*. Bernstein et al. reported higher cost of the tree algorithm when the search is limited to the vectors with proper form to break FSB [21].

In the submission to NIST, the security of FSB combined with the final compression g was evaluated with respect to the **uL** bounds in Table 5.2. For concreteness consider collision resistance of FSB_{256}, the same argument can be made for the remaining variants too. In this case the **uL** bound is 2^{153}. Output of f is 1024 bits long and this security is considered deep below the trivial **U** bound 2^{512} due to generic attacks. This discrepancy is "fixed" by the final transformation g. The 1024 bits are compressed to yield a 256-bit result using the hash function Whirlpool.

The authors remark that "the complexities of the attacks on the FSB compression function ... can thus be transposed directly to the whole hash function and are all above the complexities of generic attacks on the whole FSB ..." Collisions in $g \circ f$ are *no harder* to find than collisions in f. The **uL** bound 2^{153} thus transfers to $g \circ f$. This is above the trivial **U** bound due to a generic attack. We are therefore left with a **U** bound 2^{128}, that is trivially also a **uL** bound with respect to any **L** bound. Such a bound is independent of the hardness assumption. So is the claim of the designers that FSB_{256} achieves 128-bit collision resistance. All the variants are claimed to achieve the maximum security possible.

It might seem that the problem is that g compresses too much. What if the cost of generic attacks on $g \circ f$ is above the cost of attacks on f? Can the intermediate 1024 bits in FSB_{256} be compressed a little less to maintain some of the provable security? With a 320-bit output from g, attacking f might be faster. This would nevertheless only lead to a **U** bound, because a collision in $g \circ f$ does not imply a collision in f.

Could lower bounds be preserved? A collision in $g \circ f$ implies a collision in one of the two components. If there were **L** bounds l_f and l_g tied to f and g respectively, the smaller of the two would then be a lower bound on security of the composition. Such proof would be possible if g could be assumed collision resistant in the first place. Because g is fixed, the assumption is trivially false. Although composing the FSB compression function with a provably collision resistant final transformation can preserve the lower bound(s), it would resemble a circular argument where a provable collision resistant function is designed given a provable collision resistant function. Finiasz remarks that "... as far as provable security is concerned, choosing a provably collision resistant function g is probably the only choice at the moment" [46]. The later submission to NIST leaves this simple observation out. The authors of FSB consider collision resistance of Whirlpool too strong an assumption [7]. For eventual collisions in Whirlpool to extend to the complete FSB one needs to invert the FSB primitive. This is also an argument from an attack perspective, it does not lead to an **L** bound.

Summary

One claim of the designers in [7] reads as follows:

> The most decisive advantage of FSB is that it comes with a proof

> of reduction to hard algorithmic problems. An algorithm able to find collisions on FSB or to invert FSB is also able to solve hard problems from coding theory.

No such statement is proved in the FSB proposals. The security level claimed by designers is not supported by a type **L** bound. This is due to lack of type **L** bounds in an assumption and in particular due to the presence of the final compression by Whirlpool.

The complexity estimates in Table 5.2 were later slightly updated by Finiasz and Sendrier [49]. There the quantities were interpreted as lower bounds on the cost of algorithms solving CSD and CF. Extending our notation, the type of the bounds in that context would be **luL** (i.e. lower bound on a particular **uL** bound). Finiasz and Sendrier suggested these quantities be adopted as **L** bounds. Not considering the final transformation g, the **L** bounds would then transfer to the compression function f.

5.1.2 Really Fast Syndrome Based Hash

Bernstein et al. designed a fast variant of FSB by optimizing the matrix A and by hashing in a smaller group than any FSB variant proposed for SHA-3 [22]. The parameters used are $b = 8, k = 112, n = 509$, there is no final transformation. No provable security claims were made at all. Based on the cost of attacks, the authors claim collision resistance of at least 2^{128}.

5.2 Hash Functions from Expander Graphs

Charles et al. proposed a hash function similar to the multiplicative group design by Zémor and Tillich. At the time, the latter was not considered broken for well-chosen parameters. Messages are hashed by multiplying matrices over a finite field [35]. The group used is $G = PSL(2, \mathbb{F}_p)$ for prime $p \equiv 1 \mod 4$. It consists of all the 2×2 matrices over $\mathbb{F}_p$ with determinant 1, modulo the equivalence relation $M \sim -M$.

Let $l \equiv 1 \mod 4$ be a prime number smaller than p, such that l is a quadratic residue modulo p. Let $\mathbf{i}$ be an integer such that $\mathbf{i}^2 \equiv -1 \pmod{p}$. Define the set of $k = l + 1$ generators $A = \{a_1, \ldots, a_{l+1}\}$ where

$$a_j = \begin{pmatrix} g_0 + \mathbf{i}g_1 & g_2 + \mathbf{i}g_3 \\ -g_2 + \mathbf{i}g_3 & g_0 - \mathbf{i}g_1 \end{pmatrix}$$

and (g_0, g_1, g_2, g_3) cycle through all the $l+1$ integer solutions to $g_0^2+g_1^2+g_2^2+g_3^2 = l$ with odd $g_0 > 0$ and even g_1, g_2, g_3. The choice of G and A is inspired by an expander graph construction by Lubotzky, Phillips and Sarnak [83]. The function is therefore referred to as *LPS Hash*.

The construction follows along the lines of Bosset's proposal (cf. Section 4.1). Messages are processed in blocks of $b = \lfloor \log_2 (|A| - 1) \rfloor$ bits. A fixed injective encoding $\phi : \{0,1\}^b \to A$ would allow trivial collisions, because A is closed under inversion. Charles et al. propose to encode input blocks adaptively in order to avoid the element a_i to be immediately followed by a_i^{-1}. Fix injective functions $\phi_j : \{0,1\}^b \to A \setminus \{a_j\}$ for $j = 1$ to $l + 1$. The function ϕ_j leaves out the element a_j. If the i-th block of input $m[i]$ was mapped to a_j, apply the function ϕ_j to encode the block $m[i + 1]$. Use an arbitrary (fixed) ϕ_j to encode the first block.

Morgenstern Hash Petit et al. generalized the above construction to compute over any finite field and proposed to use binary fields for efficiency [106].

Due to the increase in b, the new functions perform fewer matrix multiplications per input bit than the function by Zémor and Tillich. The families also provide unconditional separation of colliding inputs. The property is connected to a lower bound on the length of cycles in the corresponding Cayley graph.

5.2.1 Provable Security

The LPS and Morgenstern hash functions had originally been claimed to be *provable collision resistant* or *provable*, respectively. The underlying hard computational problem is the search for a product $\sum_{i=1}^{N} a_{\sigma(i)}^{e_i} = 1_G$ for $\sigma : \{1, \ldots, N\} \to \{1, k\}$ under the condition that $a_{\sigma(i)} a_{\sigma(i+1)} \neq 1_G$ and that $\sum |e_i|$ is within $O(\log p)$. The last condition rules out the useless "trivial" solutions that raise an a_i to very high powers such as $|G|$. Such messages are too long to be relevant for security.

The implicit security proofs were not used to derive quantified type **L** bounds on collision resistance. Neither were any **U** bounds provided, apart from the trivial **U** bounds due to generic attacks. Provable security was understood to be established simply by linking it to the group problem. There was no quantified estimate on the hardness the underlying problem.

Parameters should be selected such that p is of "cryptographic size". As an example it was suggested that $\log_2 p \approx 1024$ and $l = 5$.

5.2.2 Cryptanalysis

Tillich and Zémor found an efficient collision attack on LPS by lifting the computations to the group $PSL(2, Z[i])$ [140]. Collisions for a variant with 1024-bit p were computed on a single workstation within seconds. The method was adapted to the Morgenstern hash and extended to find preimages by Petit et al. [107]. The attacks exploit the special form of the generators and do not solve the group problem in general.

5.3 Generalized Compact Knapsacks

Micciancio proposed a family of compression functions that generalize group knapsacks to finite commutative rings [93, 94]. Given a ring R, let $A \subseteq R$ be a set of weights. Extend the set of possible coefficients in a knapsack from $\{0, 1\}$ to a possibly larger set $D \subseteq R$. For fixed $k = |A|$ let $b = \lfloor \log_2 |D| \rfloor$ and fix an injective encoding $\phi : \{0, 1\}^b \to D$. To compress bk bits, compute the function

$$H(m) = \sum_{i=1}^{k} \phi(m[i]) a_i \ ,$$

where the additions and multiplications are performed in R. Such a function performs a single ring multiplication and addition per b input bits and only k ring elements are needed to specify a particular instance. It can be viewed as a k-sum in the additive group of R.

For appropriate choices of R and D the above function was shown to be asymptotically preimage and collision resistant on average under a worst case assumption on special classes of lattices. Preimage resistance was shown by Micciancio [93]. Collision resistance of related function families was established independently by Peikert and Rosen [105] as well as by Micciancio and Lyubashevsky [85]. The reductions are similar to the famous result of Ajtai sketched in Section 3.3.3.

5.3.1 SWIFFT

A practical hash function following the above principles was proposed by Lyubashevsky et al. [86, 87].

Definition

The SWIFFT compression function is defined in the ring $R = \mathbb{Z}_q[\alpha]/\left(\alpha^d + 1\right)$ for $q = 257$ and $d = 64$. Let A be a fixed set of k weights from the ring for

$k = 16$. Restrict the set D to d-bit binary vectors interpreted as elements of R. To hash a dk bit message compute the expression

$$H(m) = \sum_{i=1}^{k} m[i]a_i \ , \tag{5.1}$$

where each $m[i]$ is interpreted as an element of D. The resulting ring element is a d-dimensional vector over $\mathbb{Z}_q$ and can be represented by about $n = d \log_2 q \approx 512$ bits.

Addition of elements of R corresponds to addition in $(Z_q)^d$. The ring products $m[i]a_i$ involve multiplication of two polynomials over $\mathbb{Z}_q$ modulo $\alpha^d + 1$. Lyubashevsky et al. significantly reduce the complexity of this step by a series of tricks involving FFT multiplication. The polynomial $m[i]$ is evaluated on all the odd powers ω^{2j-1} of $\omega = 42$ for $j = 1$ to d to obtain d primitive Fourier coefficients of $m[i]$. Because the a_i are constant, their FFT representation can be precomputed once and for all. The FFT coefficients of the product $m[i]a_i$ can then be computed in d multiplications in $\mathbb{Z}_q$. The result can safely be left in the Fourier representation, it is not necessary to compute the inverse FFT. A fixed linear bijection has no effect on the security of H.

Security

As an instance of a generalized compact knapsack, the compression function SWIFFT is proved preimage and collision resistant on average. The related worst case assumption says that short vectors are hard to approximate in the lattice

$$\mathcal{L} = \{g \mod (\alpha^d + 1) : g \in I\}$$

for an ideal $I \subseteq \mathbb{Z}[\alpha]/\left\langle \alpha^d + 1 \right\rangle$ and $d \to \infty$. The security proof is asymptotic and therefore does not directly lead to a quantified type **L** bound on security for d fixed. Such reductions are commonplace in lattice-based cryptography. We can nevertheless assume that for a particular choice of d the proof does establish type **L** bounds on security that are not explicitly quantified.

The designers of SWIFFT are careful to interpret the security reduction and make clear that they do *not* claim *full* security of 2^n or $2^{n/2}$ against preimage resp. collision attacks. Security of the instance should further be assessed by considering attacks.

Connection to Subset Sum Because $\alpha^d = -1$ in the ring R the product of polynomials $a_i m[i]$ can be represented as a matrix-vector product $A_i m[i]$

over $\mathbb{Z}_q$. The matrix A_i is a $d \times d$ skew-circulant matrix built from the coefficients of the polynomial a_i. If M denotes the $d \times kd$ matrix $A_1|A_2|\dots A_d$ and m is a kd-bit message, then the SWIFFT compression function simply computes the matrix product

$$H(m) = M \times m \ . \tag{5.2}$$

In this above representation the SWIFFT function becomes an instance of the compression function GGH (cf. Section 4.6) with a special defining matrix.

Recall that FSB also had similar representation as a matrix product. Just as it was the case for FSB, the linear system of SWIFFT is easy to invert. Not every solution corresponds to an input message. For SWIFFT, coefficients of preimages must be limited to $\{0, 1\}$ and for collisions to $\{-1, 0, 1\}$. Another similarity with FSB is the use of cyclic shifts to reduce the total size of the matrix.

SWIFFT and the Tree Algorithm Another possible interpretation of SWIFFT is as a k-sum. The d-bit chunk $m[i]$ is mapped to one of the 2^d possible elements $a_i m[i] \in (\mathbb{Z}_q)^d$. The compression function is then simply a sum of precisely 16 elements of the additive group of d-dimensional vectors modulo q. Lyubashevsky et al. analyze the cost of Wagner's tree algorithm involved in preimage and collision search. The smooth value of $d = 64$ allows enough freedom in finding the appropriate subgroups K_i needed. An inversion attack can start with 8 lists and needs 2^{128} time and space. A collision finding attack with 16 lists costs 2^{106}. These are derived as *lower* bounds on the cost of two particular attacks. In our language the type for such bounds is **lU**. These quantities can actually be considered reasonable *upper* bounds on security, in particular because the algorithm by Minder and Sinclair is likely to run faster on the instances in question. We can therefore safely consider the two quantities 2^{128} and 2^{106} to be type **U** bounds. This trivially implies that even if the provable **L** bounds were quantified, preimage resistance cannot exceed 2^{128} and collision resistance is below 2^{106}. The type **U** bounds associated to attacks provide concrete and valuable information on provable security.

The security reduction links security of SWIFFT to worst case lattice problems in dimension $d = 64$. Buchman and Lindner suggest that any lower bounds on inversion should be considered "insignificant" [29]. This reasoning is based on the results of Gama and Nguyen who claim that exact SVP in dimension up to approx. 70 should be considered easy.

SWIFFT and Lattice Reduction The designers of SWIFFT point out that the space of solutions to the system (5.2) is a modular lattice with dimension kd. A non-zero vector with entries limited to $\{-1, 0, 1\}$ reveals a pair of colliding messages for H. Such vectors are shortest in the l_∞ norm. The dimension $kd = 1024$ is well out of reach of modern lattice reduction algorithms.

5.3.2 SHA-3 Candidate

A hash function SWIFFTX based on the SWIFFT primitive was submitted as a SHA-3 candidate by Arbitman et al. [6]. The proposal was not selected for the second round.

The SWIFFTX compression function maps 2048 bits to 520 bits by combining four calls to SWIFFT, each with a different set of knapsack weights. The four calls to SWIFFT are combined in a way that is believed to make the known attacks inefficient. First the 2048-bit input is compressed using three parallel instances of SWIFFT with $k = 32$ to compute 1560 bits. A fixed invertible injection extends this intermediate result to 1600 bits. Then a single SWIFFT instance with $k = 25$ is applied and 520 bits are output.

The designers are careful to preserve the provable collision and preimage resistance of the building blocks. A collision in SWIFFTX implies a collision in at least one of the four SWIFFT components. The ability to find preimages also implies inversion of a SWIFFT building block. Effectively, the *least* of the lower bounds that applied to the building blocks is valid for SWIFFTX.

For the internal SWIFFT instance with $r = 25$ the designers mention a preimage finding attack that applies Wagner's algorithm and requires 2^{100} operations. This is a **U** bound, hence also a **uL** bound on the unknown type **L** bound tied to the proof. Therefore any type **L** bound on preimage resistance transferred from SWIFFT to SWIFFTX is *at most* 2^{100}. A collision finding attack on the instance with $r = 25$ would probably be even easier, in any case the same quantity 2^{100} is a **uL** bound on the **L** bound on collision resistance inherited from SWIFFT. The three instances with $r = 32$ are likely to be strictly easier and may drag the **uL** bounds even lower.

The designers of SWIFFTX analyzed various sorts of attacks on the complete compression function and concluded that none of them was faster than the generic attacks. The construction thus "wipes out" the type **U** bounds there were for SWIFFT. One ends up with the trivial **U** bounds due to generic attacks.

In conclusion, Arbitman et al. *do* claim *full* n-bit preimage security and

$n/2$-bit collision security. This may well be a reasonable conclusion, but it is definitely not supported by a security proof.

The **uL** bound 2^{100} on the provable security inherited from SWIFFT remains valid. This is not a complexity of an actual attack, improved proofs may be possible.

5.4 Very Smooth Hash

Very Smooth Hash introduced by Contini et al. [40] offers provable collision resistance under an assumption heuristically linked to the hardness of factoring. It can be seen as a generalization of the function $H(m) = a^m \mod M$ for fixed a and M a hard-to-factor integer. A pair of colliding inputs reveals a multiple of $\varphi(M)$ that can be used to factor M fast. The mapping was considered by Pointcheval [110] as well as Shamir and Tauman [130]. In VSH, the single element a is replaced by a larger set of weights and the function computes a modular multi-exponentiation.

5.4.1 Basic VSH

Let M be an n-bit hard to factor modulus. Denote the i-th prime number by p_i and let $p_0 = -1$. Let k be the largest integer such that $\prod_{i=1}^{k} p_i < M$. Let m be an l-bit message to be hashed, consisting of bits $m_1, \ldots, m_l$ and assume $l < 2^k$. The basic VSH algorithm runs as follows:

1. Initialize $x_0 = 1$.
2. Let $\mathcal{L} = \lceil \frac{l}{k} \rceil$. Let $m_i = 0$ for $l < i \leq \mathcal{L}k$.
3. Let $l = \sum_{i=1}^{k} l_i 2^{i-1}$ with $l_i \in \{0, 1\}$ be the binary representation of l and define $m_{\mathcal{L}k+i} = l_i$ for $1 \leq i \leq k$.
4. For $j = 0, 1, \ldots, \mathcal{L}$ in succession compute
$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_i^{m_{jk+i}} \mod M \ .$$
5. Return $x_{\mathcal{L}+1}$.

The function iteratively processes blocks of k bits in a mode similar to the Merkle-Damgård construction, the value x_j serves as a chaining variable.

Isolate the expression

$$\prod_{i=1}^{k} p_i^{m_{jk+i}} \mod M \tag{5.3}$$

from Step 4. It computes a subset product in the group $\mathbb{Z}_M^*$ on the set of weights $A = \{p_1, \ldots, p_k\}$. The number of primes entering the product can range from 0 to k. This operation mapping k bits of the message to a group element is by construction injective.

Fast VSH

The number of multiplications performed can be reduced by increasing the total number of primes. Fix a small integer $b > 1$ and use $k' = k2^b$ small prime numbers for an integer k. Denote b-bit blocks of the l-bit input message m by $m[i]$. The Fast VSH algorithm proceeds as follows:

1. Initialize $x_0 = 1$.
2. Let $\mathcal{L} = \lceil \frac{l}{bk} \rceil$. Pad the message with zero bits up to an integral multiple of bk.
3. Append a bk-bit binary representation of l to the message, denote the new chunks $m[\mathcal{L}k + 1]$ to $m[(\mathcal{L} + 1)k]$.
4. For $j = 0, 1, \ldots, \mathcal{L}$ in succession compute

$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_{(i-1)2^b + m[jk+i]+1} \mod M \ .$$

5. Return $x_{\mathcal{L}+1}$.

One choice of k was the maximal integer such that $\prod_{i=1}^{k} p_{i2^b} < M$, larger values of k are also possible. The hash function processes bk bits per iteration, the operation

$$\prod_{i=1}^{k} p_{(i-1)2^b + m[jk+i]+1} \mod M \tag{5.4}$$

in Step 4 can be seen as a group k-product. The number of primes forming the product always equals k, the operation is injective by construction.

Security Proof

Call an integer *very smooth* if all its prime factors are bounded by $(\log M)^c$ for a fixed constant c. Contini et al. prove VSH collision resistant if the following problem is hard:

Problem 5.1 (VSSR: Very Smooth number nontrivial modular Square Root)**.** *Let M be the product of two primes of approximately the same size and let $k' \leq (\log M)^c$. Given M, find x such that $x^2 \equiv \prod_{i=0}^{k'} p_i^{e_i} \mod M$ and at least one of the $e_0, e_1, \ldots, e_{k'}$ is odd.*

Theorem 5.1. *Collision search for VSH is at least as hard as solving VSSR with $k' = k$.*

We include the original proof by Contini et al. [40].

Proof. We show that different colliding messages m and m' lead to a solution of VSSR. Let $x'_{\ldots}$ denote the $x_{\ldots}$ values in the VSH algorithm applied to m' and let $l, \mathcal{L}$ and $l', \mathcal{L}'$ be the bitlengths and number of blocks of m and m', respectively. Since m and m' collide, $m \neq m'$ and $x_{\mathcal{L}+1} = x'_{\mathcal{L}'+1}$.

First consider the case of $l = l'$. Let $m[j]$ denote the m's jth k-bit block, $m[j] = (m_{j\cdot k+i})_{i=1}^{k}$, and let $t \leq \mathcal{L}$ be the largest index such that $(x_t, m[t]) \neq (x'_t, m'[t])$ but $(x_j, m[j] = (x'_j, m'[j])$ for $t < j \leq \mathcal{L} + 1$. Then,

$$(x_t)^2 \times \prod_{i=1}^{k} p_i^{m_{t\cdot k+i}} \equiv (x'_t)^2 \times \prod_{i=1}^{k} p_i^{m'_{t\cdot k+i}} \mod M \ . \tag{5.5}$$

Let $\Delta = \{i : m_{t\cdot k+i} \neq m'_{t\cdot k+i}, 1 \leq i \leq k\}$ and $\Delta_{10} = \{i \in \{1, \ldots, k\} : m_{t\cdot k+i} = 1 \text{ and } m'_{t\cdot k+i} = 0\}$. Because all factors in Equation (5.5) are invertible modulo M, it is equivalent to

$$\left[(x_t/x'_t) \times \prod_{i\in\Delta_{10}} p_i\right]^2 \equiv \prod_{i\in\Delta} p_i \mod M \ . \tag{5.6}$$

If $\Delta \neq 0$, Equation (5.6) solves VSSR. If $\Delta = 0$, then $(x_t)^2 \equiv (x'_t)^2 \mod M$ and $t \geq 1$ (since $m \neq m'$ and using the definition of t). With $x_t \not\equiv \pm x'_t \mod M$ VSSR can be solved by factoring M. If $x_t \equiv \pm x'_t \mod M$ then $x_t \equiv -x'_t \mod M$ leads to $(x_{t-1}/x'_{t-1})^2$ being congruent to -1 times a very smooth number and thus solves VSSR.

Now consider the case $l \neq l'$. Since $x_{\mathcal{L}+1} = x'_{\mathcal{L}'+1}$, we have $(x_{\mathcal{L}}/x'_{\mathcal{L}'})^2 \equiv \prod_{i=1}^{k} p_i^{l'_i - l_i} \mod M$. Since $|l'_i - l_i| = 1$ for at least one i, VSSR is solved using a transformation as in Equation (5.6). □

The proof can be adapted to the Fast VSH setting to show that a collision solves VSSR for $k' = k2^b$.

Computational VSSR Assumption Solving VSSR is as hard as factoring a hard to factor n'-bit modulus, where n' is the least positive integer such that

$$L'[2^{n'}] \geq \frac{L'[M]}{k'} \ , \tag{5.7}$$

and the function

$$L'[M] = e^{1.923(\log M)^{1/3}(\log\log M)^{2/3}} \tag{5.8}$$

approximates the running time of Number Field Sieve factoring the integer M. Here k' stands for the total number of primes used by the hash function, that is equal to k or $k2^b$ for Basic and Fast VSH, respectively.

To justify the assumption, observe that a solution to VSSR is precisely the kind of relation collected in modern factoring algorithms (cf. Section 3.1, p. 17). Heuristically, given more than $k'+1$ solutions to VSSR, one can use linear algebra to find $X \not\equiv \pm Y \mod M$ such that $X^2 \equiv Y^2 \mod M$ and expect to factor the modulus.

If NFS is assumed to be the fastest method for factoring integers of the form of M, the VSSR problem can heuristically be assumed no easier than a fraction $1/k'$ of the cost of NFS. The above assumption can directly be used to derive a provable lower bound on collision resistance for a particular Fast VSH variant.

The requirement $\prod_{i=1}^{k} p_i < M$ resp. $\prod_{i=1}^{k} p_{i2^b} < M$ on k can be relaxed at the cost of increased k' and in consequence a possible drop in security. Although the k-sums (5.3) and (5.4) are no longer injective, Theorem 5.1 remains valid. Removal of the restriction on k allows to process more bits per iteration, and speeds the function up. The drop in security can be compensated for by selecting a longer modulus.

Table 5.3 lists the VSH variants proposed by Contini et al. as well as the associated **L** bounds on collision resistance. The complexity is expressed in units different from the usual “bit security” of traditional hash functions. For example, a collision in Basic VSH with $n = 1516, k = 152$ and $b = 8$ is at least as hard to find as it is to factor a 1024 bit RSA modulus.

Table 5.3: Original VSH Variants and Security Levels

n'	Method	n	k	b	k'
1024	Basic	1234	152	–	–
1024	Basic	1318	1024	–	–
1024	Fast	1516	256	8	65536
2048	Basic	2398	272	–	–
2048	Basic	2486	1024	–	–
2048	Fast	2874	1024	8	262144

Generating Collisions

No attacks that would achieve the above lower bounds are known. We describe the fastest known attack and the corresponding non-trivial type **U** bound on collision resistance. Collisions turn out to be trivial to create if $\varphi(M)$ is known. Computing $\varphi(M)$ from M is as hard as factoring the modulus. Observe that the output of H equals

$$\prod_{\alpha=1}^{k'} p_\alpha^{e_\alpha} \mod M \ , \tag{5.9}$$

where the exponents e_α are integers $\mathcal{L}+1$ bits long derived from the input. The product (5.9) does not change if an integral multiple of $\varphi(M)$ is added to any of the exponents. In Basic VSH $k' = k$ and the j-th most significant bit of e_i is equal to the bit $m_{(j-1)k+i}$, i.e. $e_i = \sum_{j=1}^{\mathcal{L}+1} m_{(j-1)k+i} 2^{\mathcal{L}+1-j}$. Every set of non-negative exponents corresponds to some input message. Collisions as well as second preimages are easy to create for basic VSH if $\varphi(M)$ is known.

For Fast VSH, the b-bit input block $m[jk+i]$ sets the j-th most significant bit of the exponent $e_{(i-1)2^b+m[jk+i]+1}$ to 1. The k-product 5.4 is always composed of precisely k primes, one from among $p_{(i-1)2^b+1}, \ldots, p_{i2^b}$ for each $1 \leq i \leq k$. A change in any single of the exponents $e_{(i-1)2^b+1}, \ldots e_{i2^b}$ for some i must be compensated for by appropriate changes in other exponents. There is less freedom than in Basic VSH, but collisions are easy to find in this setting, too.

The collisions created in this way are quite long, at least one of the messages has nk or nkb bits for Basic and Fast VSH, respectively. Creating short collisions appears to be a much harder problem, even if $\varphi(M)$ is known.

The algorithm sketched above can find collisions in Basic or Fast VSH in time $L'[M]$. This is a **U** bound on collision resistance. There is a gap between the proved type **L** bound and the least known type **U** bound, the two quantities are a factor of k' apart. So far, no result has appeared that would get the (provable) lower bound closer to the complexity of factoring the modulus M. Collision search may even be harder than solving VSSR, Theorem 5.1 only proves that it is no easier. A single random VSSR solution with x does not in general reveal colliding inputs for VSH. Only with some $k' + 2$ solutions to VSSR one can factor the modulus and form colliding messages freely.

5.4.2 Discrete Logarithm Variant

The factorization of M is essentially a trapdoor in the function. Contini et al. proposed a related compression function that has no such trapdoor. If the modulus is replaced by an n-bit prime number $p = 2q + 1$ for prime q, one obtains the function VSH-DL. The security is then related to hardness of discrete logarithms modulo p. Select an integer block length k. The compression function VSH-DL maps $\mathcal{L}k$ bits to n bits for fixed $\mathcal{L} \leq n - 2$ such that $\mathcal{L}k > n$. It uses the same iteration as VSH. Because the length of input is fixed, it is not necessary to append the encoding of the length. To hash $\mathcal{L}k$ bits proceed as follows:

1. Initialize $x_0 = 1$.
2. For $j = 0, 1, \ldots, \mathcal{L} - 1$ in succession compute

$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_i^{m[jk+i]} \mod p \ .$$

3. Return $x_{\mathcal{L}}$.

The restriction $\mathcal{L} \leq n - 2$ prevents an analogy to the collision attack we described for VSH. The exponents e_α formed from the input are too short to allow modification by a non-zero multiple of $\varphi(p)$. The length of the latter number is $n - 1$ bits.

Fast VSH-DL

For completeness, let us describe the Fast variant of VSH-DL. The function maps $\mathcal{L}bk$ bits to n bits for fixed $\mathcal{L} \leq n - 2$ such that $\mathcal{L}bk > n$ as follows:

1. Initialize $x_0 = 1$.

2. For $j = 0, 1, \ldots, \mathcal{L} - 1$ in succession compute

$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_{(i-1)2^b + m[jk+i]+1} \mod p \ .$$

3. Return $x_{\mathcal{L}}$.

Security Proof

Contini et al. reduce collision resistance of the above function to the following computational problem:

Definition 5.1 (VSDL: Very Smooth number Discrete Log)**.** *Let p, q be prime numbers with $p = 2q + 1$ and let $k' \leq (\log p)^c$. Given p, find integers $e_1, e_2, \ldots, e_{k'}$ such that $2^{e_1} \equiv \prod_{i=2}^{k'} p_i^{e_i} \mod p$ with $|e_i| < q$ for $i = 1, 2, \ldots, k'$, and at least one of $e_1, e_2, \ldots, e_{k'}$ is non-zero.*

Theorem 5.2. *A collision in VSH-DL solves VSDL for $k' = k$.*

Proof. If $e_i = \sum_{j=1}^{\mathcal{L}} m_{(j-1)k+i} 2^{\mathcal{L}-j}$ for $1 \leq i \leq k$, then the function simply computes the product $\prod_{i=1}^{k} p_i^{e_i}$. A collision $m, m' \in \{0,1\}^{\mathcal{L}k}$ such that $m \neq m'$ leads to the congruence $\prod_{i=1}^{k} p_i^{e_i} \equiv \prod_{i=1}^{k} p_i^{e_i'} \mod p$ where the exponents e_i' correspond to m'. Rearranging this congruence, a solution $2^{e_1 - e_1'} \equiv \prod_{i=2}^{k} p_i^{e_i - e_i'} \mod p$ follows, because $|e_i' - e_i| < 2^{\mathcal{L}} \leq 2^{n-2} \leq q$ for all i and $e_i' - e_i \neq 0$ for some i since $m \neq m'$. □

The theorem extends to Fast VSH-DL with $k' = k2^b$. No computational VSDL assumption was made in the proposal by Contini et al. An easy analogy with VSSR and factoring seems not to be available. Collecting k' solutions to VSDL allows to recover the logarithms of the k' small prime numbers. The primes form too small a basis to allow computation of arbitrary logarithms in $\mathbb{Z}_p^*$. No computational lower bound on collision resistance of VSH-DL can immediately be derived.

Generating Collisions

In contrast to Basic VSH, any single solution to VSDL immediately leads to a pair of colliding messages in VSH-DL. The ability to compute discrete logarithms modulo p leads to a straightforward way to create collisions and even second preimages in VSH-DL if $\mathcal{L}$ reaches the maximal value $n - 2$.

The case of Fast VSH-DL needs a little bit more care to balance the exponents, but collisions can be computed there as well. For these variants, there is a type **U** bound on collision resistance with cost corresponding to DL computation. The attacks appear to fail for $\mathcal{L}$ significantly smaller than $n-2$. The complexity of the problem is unclear, a non-trivial **U** bound is not immediate.

5.4.3 Preimage Resistance

The designers of VSH made no claims on preimage resistance of any variant. Saarinen pointed out that digests of messages with known length l bits can be inverted with only $2^{l/2}$ effort thanks to the multiplicative structure [120]. His attack is essentially an instance of the 2-list algorithm described in Section 3.6.1.

We can generalize this to an algorithm that can find preimages under VSH in significantly less than $2^{n/2}$ operations. The (non-DL) VSH hash function can be interpreted as a very special case of MuHASH. If we can compute the additive representation of the k' prime numbers in $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{q-1}, +)$, we can run the $O(2^{2\sqrt{n}})$ tree algorithm to find preimages as in the case of AdHash. Given $L_M[1/3, c]$ algorithms for factoring and discrete logarithms, the running time is asymptotically dominated by the cost of the tree algorithm and becomes $O(2^{2\sqrt{n}})$. The attack involves more effort than factoring the modulus M, it is therefore no threat to any of the functions listed in Table 5.3.

5.4.4 The Role of Small Primes

Small prime numbers used in VSH have two major positive effects. Because a particular function instance is defined by $k2^b$ group elements, if full n bits were used for every single entry, the memory requirements would soon become prohibitive.

Another advantage of the use of small primes is speed. Multiplication of the n-bit modulus by a small prime is much easier an operation than a full $n \times n$ bit multiplication. The small prime numbers are absolutely necessary for VSH to remain practical.

The fastest variant of VSH proposed is Fast VSH with $n = 1536, k = 256$ and $b = 8$. Contini et al. reported that it ran at the speed 1.1 MB/s on a 1GHz Pentium III, that is approx. 900 cycles/byte. This was at the time about 25 times slower than SHA-1. Although VSH brought improved performance compared to earlier collision resistant functions related to factoring, the speed was very slow compared to “classical” hash functions.

5.4.5 Extensions to Other Groups

Lenstra et al. generalized the concept of VSH-DL to other groups where discrete logarithms are hard. The output of these variants can be made shorter than in VSH based on modular multiplication. This is achieved by selecting groups where the DL problem is relatively harder. One such group is a multiplicative cyclotomic subgroup in a degree six extension field. Another example given was a group of points on an elliptic curve. A VSH variant in the latter group was claimed to provably achieve $2^{n/2}$ collision resistance for n-bit output. The compression functions in both group classes were reported to be several times slower than multiplicative VSH with equivalent security [79]. One of the reasons for this is the absence of an equivalent of the small prime numbers in $\mathbb{Z}_M^*$, i.e. elements that allow both fast multiplication and compact representation.

Chapter 6

New VSH Variants

This chapter is an extended version of the article *Faster and Smoother – VSH Revisited* [123].

In the previous chapter we considered the tree algorithm in preimage search for VSH. A similar approach can be used to find collisions. It had never been considered before for VSH, most likely because the tree algorithm cannot easily be applied without factoring the modulus first. It would therefore pose absolutely no threat to collision resistance of original VSH, that security property can be violated by factoring M anyway.

The collision attack on VSH sketched in the previous chapter depends in a crucial way on the fact that the exponents in the expression $\prod_{\alpha=1}^{k'} p_{\alpha}^{e_{\alpha}} \bmod M$ computed within VSH can grow without limit. The VSH-DL design adds a minor restriction by asking the exponents be at least one bit shorter than the binary expansion of the group order. In this chapter we design new variants of VSH that impose a much stricter restriction there and limit all the exponents to one. We show that the change preserves the provable connection to VSSR or VSDL for appropriate moduli.

6.1 A Variant Without Modular Squaring

Consider Fast VSH as described in Section 5.4.1. Observe that if $kb > n$, the k-sum operation

$$\prod_{i=1}^{k} p_{(i-1)2^b+m[jk+i]+1} \mod M$$

from Step 4 of Fast VSH compresses the input. It can therefore be extended to a hash function in the usual ways, such as the Merkle-Damgård mode.

Our new Fast VSH variant will impose the condition $kb > n$ and build a compression function.

Faster VSH

Have an n-bit modulus M, let k, b be integers such that $kb > n$ and $k2^b < \log(M)^c$. Let M be coprime to all the prime numbers $p_1, \ldots, p_{k2^b}$. Define a compression function H from kb bits to n bits that outputs

$$H(m) = \prod_{i=1}^{k} p_{(i-1)2^b+m[i]+1} \mod M \ . \tag{6.1}$$

The function computes a modular product of precisely k out of the primes $p_1, \ldots, p_{k2^b}$. Let $\phi_i : \{0,1\}^b \to \mathbb{Z}_M^*$ for $1 \leq i \leq k$ be a function that on input x returns the prime $p_{(i-1)2^b+x+1}$ where x is interpreted as an integer less than 2^b. The computation of Faster VSH in (6.1) can equivalently be described as the product

$$\prod_{i=1}^{k} \phi_i(m[i]) \tag{6.2}$$

computed in the multiplicative group $\mathbb{Z}_M^*$.

Collision search in Faster VSH modulo a product of two primes can be reduced to the same hard problem as the original VSH.

Theorem 6.1. *A collision in Faster VSH with $M = pq$ a hard to factor integer solves VSSR for $k' = k2^b$.*

Proof. Let $m \neq m'$ be two bk-bit messages such that $H(m) = H(m')$ and

$$H(m) = \prod_{i=1}^{k} \phi_i(m[i]) \mod M \tag{6.3}$$

$$H(m') = \prod_{i=1}^{k} \phi_i(m'[i]) \mod M \ . \tag{6.4}$$

Denote the computed hash value $H(m)$ by x. From (6.3) and (6.4) it follows that

$$x^2 \equiv \prod_{i=1}^{k} \phi_i(m[i]) \times \prod_{i=1}^{k} \phi_i(m'[i]) \mod M \ .$$

Because $m \neq m'$, there exists an index $1 \leq j \leq k$ such that $m'[i] \neq m[i]$. The exponent of $\phi_i(m[i])$ on the right hand side then equals one and the above expression solves VSSR. □

The modular squaring in VSH is no more necessary for the security proof. In an analogy to the VSH-DL proof, collision search for Faster VSH modulo $M = 2q + 1$ for prime q can be reduced to the VDSL Problem.

Theorem 6.2. *A collision in Faster VSH-DL solves VSDL for $k' = k2^b$.*

With appropriate moduli, the modified function is secure under the same assumptions as original VSH, the type **L** bounds carry over. Observe that our modification now limits all the exponents in the prime products to one. The factoring attack on VSH and the discrete log attack on Fast VSH-DL no longer apply. The non-trivial type **U** bounds are not necessarily valid in the new setting.

Performance

The change to VSH proposed above slows the functions down. The original Fast VSH could process bk "fresh" bits per iteration. The new function requires a domain extender such as the Merkle-Damgård mode and needs to hash the chaining variable with every block. Only $bk - n$ message bits can therefore be processed per iteration. The performance of the modified variant is approximately a fraction $1 - \frac{n}{bk}$ of the original. The greater the compression ratio bk/n of the new function, the less significant the slowdown.

The proposed modification to Fast VSH does not appear to be useful at all. The mere fact that the original collision-finding attacks do not work might not be worth the performance loss. The type **L** bounds on collision resistance implied by the security proofs are the same after all.

As we will show in the next section, the modification allows a radical change in security assessment and in the end permits faster and/or more secure hashing.

6.1.1 The Extended Tree Algorithm

To analyze security of the new function, consider it a group k-product as in (6.2). Let L_i for $1 \leq i \leq k$ be a list that contains the 2^b primes $p_{(i-1)2^b+1}, \ldots, p_{i2^b}$. The function $\phi_i : \{0,1\}^b \to L_i$ then simply maps the integer $m[i]$ to the element on position $m[i] + 1$ in the list L_i.

Preimage search Inversion of Faster VSH is a k-sum problem in the multiplicative group $G = \mathbb{Z}_M^*$. Several other functions fit this description with appropriate G, in particular FSB and SWIFFT. The attack method of choice for such functions is the tree algorithm.

To assess preimage resistance of our new function, consider the complexity of the extended k-tree algorithm by Minder and Sinclair. The algorithm starts with k lists. If we aim for a single solution, then by the formulas on p. 29 the maximum length of a list is

$$u = \frac{n - b2^t}{\log_2 k - t} \tag{6.5}$$

for t the least integer such that

$$n \leq (\log_2 k - t + 1)b2^t \ . \tag{6.6}$$

We are interested in a lower bound on the length of a list. Replace the integer t by the exact solution to (6.6), i.e. let $t \in \mathbb{R}$ satisfy

$$n = (\log_2 k - t + 1)b2^t \ . \tag{6.7}$$

An equivalent simplification of the cost was performed by Finiasz et al. in the recent assessment of FSB security [48]. When such a t is substituted in the expression (6.5), one obtains

$$2^u = 2^{\frac{n}{\log_2 k - t + 1}} \ . \tag{6.8}$$

If $r = bk/n$ is the compression ratio of our function and $v = \log_2 k - t + 1$ is the denominator from (6.8), then the expression (6.7) simplifies to

$$\frac{2^{v-1}}{v} = r \ . \tag{6.9}$$

Collision Search We will show that collision search also corresponds to a k-sum problem. Given the k lists L_i, form new lists L_i' containing all the elements gh^{-1} for $g, h \in L_i$. Size of L_i' is approximately 2^{2b}. A collision in H corresponds to a solution to the k-sum problem with the lists L_i' and target value 1_G. A solution to the new k-sum problem leads in turn to a pair of colliding messages. A solution is necessarily of the form $g_1h_1^{-1} * g_2h_2^{-1} * \ldots * g_kh_k^{-1}$. If $m[i] \in \{0,1\}^b$ is the unique value such that $\phi_i(m[i]) = g_i$ and $m'[i] \in \{0,1\}^b$ is such that $\phi_i(m'[i]) = h_i$, then the concatenation of $m[i]$ collides with the concatenation of $m'[i]$. This leads to a collision if $m \neq m'$, or equivalently if the solution to the k-list problem does not select 1_G in all of the lists. If looking for collisions this way, we may simply remove the element 1_G from all the lists and limit the search to colliding messages that differ in all the b-bit blocks. For collision search, there will be k initial list each with length 2^{2b}, in a direct analogy to the analysis for preimages, the maximum list length in the course of the extended tree algorithm will be at least

$$2^{u'} = 2^{\frac{n}{v'}} \tag{6.10}$$

for v' a solution to

$$\frac{2^{v'-1}}{v'} = 2r \ . \tag{6.11}$$

This suggests that when the tree algorithm is applied, collision search for H is about as hard as preimage search for a similar function over the same group that compresses twice as much as H.

Our analysis assumes the k lists L_i contain random elements of the group G. This is not at all the case. We deliberately ignore this and expect the tree algorithm to behave as if the elements were random. Similar reasoning was used in most of the previous applications of the tree algorithm to hash functions, notably in the cases of FSB and SWIFFT.

In most of the applications of the tree algorithm the cost is usually measured as the product of maximum list length 2^u and the total number of lists k. In our case the approximate cost would be $k \times 2^u$ for preimage search, resp. $k \times 2^{u'}$ for collision search. The quantity is a reasonable approximation of the number of group operations performed by the tree algorithm. A single evaluation of a k-sum function H needs precisely $k-1$ group operations. The cost of preimage or collision search counted in equivalent of H evaluations is then approximated simply by the list length 2^u, resp. $2^{u'}$. This conservative estimate corresponds to our goal to bound the cost of the attacks from below.

We make several other simplifications, in particular we assume that the structure of G allows the extended tree algorithm to run with an optimal number of lists. If this is not the case, the lower bound on cost of the tree algorithm remains valid. In practice, to apply the attack in the multiplicative groups used in Faster VSH, one would first need to map all the computation to the isomorphic additive group. This may involve factoring and/or DL

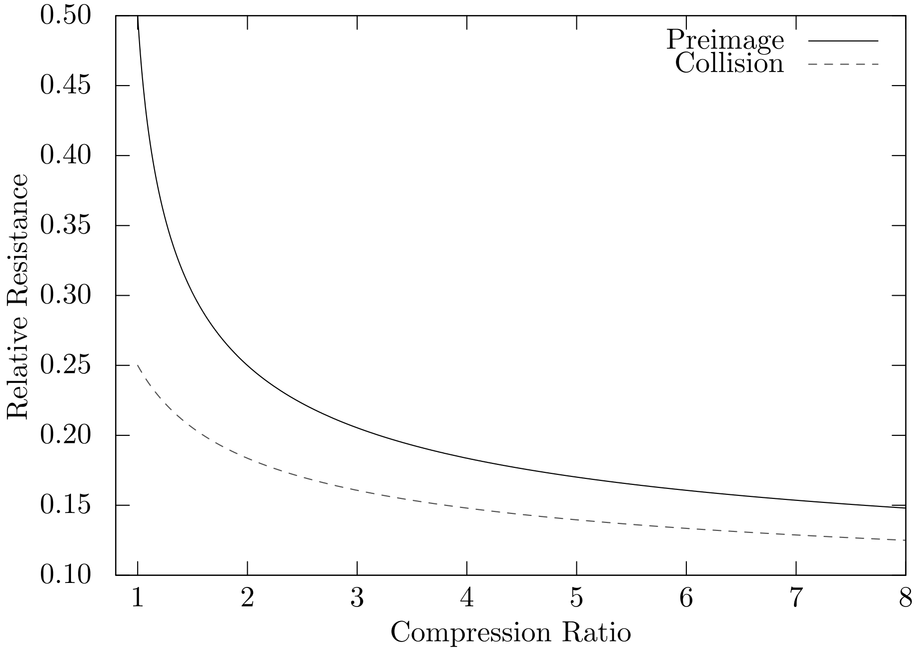

Figure 6.1: Relative Security of k-sum Functions

computations. For all the compression functions considered in this chapter, such extra cost turns out to be negligible compared to that of the tree algorithm and will be neglected.

Because $u = 2^{n/v}$, the quantity $1/v$ corresponds to the relative drop in bit-security of H. By (6.9) and (6.11) the denominator only depends on the compression ratio r. All the simplifications above allow simple estimates of the power of the tree algorithm. Figure 6.1 displays the relative bit security, i.e. $1/v$ plotted against the compression ratio r.

The bounds derived in the preceding paragraphs measure the cost of particular attacks from below and are therefore type **lU** bounds on security.

6.1.2 Security of Faster VSH

Two immediate parameter sets for Faster VSH are those originally proposed by Contini et a., provided $bk > n$. Table 6.1 captures the collision resistance level implied by the computational VSSR assumption ("factoring hardness") and the new security estimates based on the extended k-tree algorithm for

Table 6.1: Security Estimates for Variants of Faster VSH

k	n	b	Factoring Coll	Coll	Pre
256	1516	8	1024 bits	2^{326}	2^{499}
1024	2874	8	2048 bits	2^{469}	2^{603}

two such variants of Faster VSH. The modulus is assumed to be a product of two large primes. The factoring hardness measure is the complexity of factoring a hard to factor integer of bit length n' from the Computational VSSR Assumption. Columns 5 and 6 are computed using (6.10) and (6.8).

Note that the two security measures are in somewhat incompatible units. The exponent in column 5 means "bit security", the number in column 4 is "RSA security".

Hardness of factoring a 1024-bit RSA modulus is considered equivalent to roughly "80-bit" security. If Faster VSH with $n = 1516$ is assessed as a k-sum problem, it provides collision resistance of at least 326 bits. For these parameters, the removal of squarings and the use of Merkle-Damgård mode slows down the original Fast VSH by approximately 74%. This is only a moderate slowdown given the potential increase in security.

The slowdown is only approx. 35% for the other variant with $n = 2874$, while the gap between the hardness of factoring an 2048-bit modulus and 472-bit security is even more significant.

Collision resistance of 326 or 469 bits is more than enough for many years to come. We can therefore aim for lower security levels and tweak the parameters, in particular the digest length, to gain performance. Precise parameters for practical instances of the functions as well as speed measurements are given in Section 6.3.

6.2 A Variant Without Modular Reduction

The **lU** bounds on security due to the extended tree algorithm only depend on the output length n and the compression ratio r. While the structure of the particular group does affect the practicality of the algorithm to an extent, it almost certainly makes the job harder than what our estimates suggest.

Because we choose not to rely on the group structure, we propose to replace the modulus in Faster VSH by a power of two. This means that no costly modular reductions are needed in the computation of the compression function. Reduction modulo a power of two can be done essentially for free. This minor change will lead to a considerable speed-up while maintaining the k-list security.

Note that the prime $2 = p_1$ cannot be used in this setting, because it does not belong to the multiplicative group of integers modulo 2^n. The lists L_i should be filled with small primes starting with $p_2 = 3$. To compress bk input bits, compute the following product:

$$H(m) = \prod_{i=1}^{k} p_{(i-1)2^b+m[i]+2} \mod 2^n \ .$$

Any hash value will be an odd number, i.e. the least significant bit of output is always 1. Therefore only the $n-1$ leftmost bits of the n-bit modular product should be output. We will call the function *Smoother VSH* for it uses a smooth modulus in contrast to the original VSH.

Technically, the provable connection between the security of the function and the VSSR assumption (or a variant of VSDL) is preserved. This provides little confidence in security, because factoring the number 2^n is trivial. Discrete logarithms in the large cyclic subgroup of $\mathbb{Z}_{2^n}^*$ are also trivial, because the order is a power of 2. The **lU** bounds derived in Section 6.1.1 do however hold for Smoother VSH.

The original VSH as well as Faster VSH can be thought of as large families of functions even once k, n and b are fixed. One still has to select a modulus M. In the case of Smoother VSH, the modulus is fixed. Therefore there is only one instance of the function for every choice of k, n, b. There are however many ways to permute the $k2^b$ primes that lead to different compression functions.

We leave the function fixed for the moment. There is no convenient well-established hardness assumption to derive an **L** bound from. The **lU** bound we have established so far makes perfect sense even for a fixed function.

6.3 Experimental Results

We propose seven parameter sets with varying security, speed, and memory requirements. All the parameter choices were tailored to meet one of the three collision resistance levels 2^{128}, 2^{192} and 2^{256}. The preimage and collision resistance are measured as explained in Section 6.1.2. The length of the modulus is selected slightly below or equal to an integral multiple of the common word size 64 bits that is used in our test environment. Some variants, in particular Smoother VSH, may have slightly shorter output than n bits. We deliberately neglect minor differences in output length for the ease of presentation. All the variants with output length within a few bits from n are considered equivalent for the purposes of comparisons and security assessment based on the tree algorithm.

The constant b can have significant impact on performance. Small b results in many multiplications, but keeps the memory requirements down. With larger b one saves on multiplications, but needs much more memory. The value $b = 8$ used in all our variants is the most convenient choice from an implementation point of view.

6.3.1 Implementation

The modified functions are relatively efficient in software. They are also conceptually simpler compared to VSH, as there are no modular squarings to perform. All there is to do is modular multiplication. In this section we sketch how the test implementation works.

The functions were implemented in C assuming a 64-bit PC architecture, the GNU MP library is used to perform arithmetic on long integers. For simplicity, the prime numbers needed were precomputed and hardcoded in the program. All our variants are selected such that the largest prime ever needed is at most 21 bits long. This allows to use 64-bit multiplier efficiently by processing the small prime numbers in triples. Let $l = \lceil k/3 \rceil$ and $d = \lceil n/64 \rceil$. Compute the compression function as follows:

1. Determine the k primes to multiply from the input.
2. Transform the k primes to a list of l words a_i such that each a_i is the product of three p_j.
3. Initialize a $d+1$-word intermediate value x with a_1.
4. For $i = 2$ to $d - 1$ multiply x by a_i, no modular reductions are needed here.
5. For $i = d$ to l multiply x by a_i modulo M.
6. Output x.

For Smoother VSH, modular reduction in Step 5 simply discards the most significant word of the product $x \times a_i$, in practice the word will not be computed at all.

The Faster VSH code needs to reduce modulo M. Instead of dividing by M, perform a variant of Montgomery reduction to improve performance [97]. Increase x by a multiple y of M such that $x + y$ is divisible by 2^{64}. Then perform a right shift by 64 bits. The appropriate multiple can be computed as $y = z \times M$ where $z = x_0 \times (-M)^{-1} \mod 2^{64}$ for x_0 the least significant word of x. The value $(-M)^{-1} \mod 2^{64}$ can be precomputed. A final subtraction may be needed after the right shift to normalize the result below M.

A single Montgomery reduction actually divides x by 2^{64} modulo M. Step 6.5 is executed a constant number of times, therefore the resulting value of x is off by a fixed power of 2^{64}. Because a fixed bijection has no effect on the security of our compression function, we do not fix the result in the end at all.

Fast Multiplication Methods The above algorithm uses quadratic-time schoolbook multiplication. Although asymptotically faster methods exist, they appear not to help in this setting. By the GNU MP benchmark, Karatsuba multiplication becomes faster on our architecture once the operands are at least 18 words long. Moreover, it would require long arguments of comparable size and most of our inputs are short. Building longer intermediate products to benefit from Karatsuba multiplication is not worth the effort in our setting. Such an approach would be asymptotically superior, but the simple schoolbook method is empirically better for the parameter ranges considered.

6.3.2 Speed Measurements

The speed was measured on a 2.7 GHz Intel Core i7-2620M (Sandy Bridge) CPU running a 64-bit GNU/Linux system. Table 6.2 displays speed measurements for Faster VSH variants running in the Merkle-Damgård mode. Performance of Smoother VSH is summarized in Table 6.3. The tables also capture the total size of the lists of small prime numbers, represented by three bytes each. The bounds on collision and preimage resistance were computed using (6.10) and (6.8). For comparison, in the case of Faster VSH modulo a product of two primes, we include “factoring” security levels implied by the Computational VSSR assumption derived using formulas (5.7) and (5.8) in Chapter 5.

Use of the modulus 2^n makes Smoother VSH approximately twice as fast as in the case of a random n-bit modulus M. For comparison, the speed of SHA-256 on the same platform is approximately 155 MB/s or 17.4 cycles

Table 6.2: Faster VSH Variants

k	n	Coll	Pre	Factoring Coll	Memory	MB/s	Cycles/b
128	640	2^{128}	2^{184}	375	96 kB	36.4	74.0
256	768	2^{128}	2^{166}	452	192 kB	47.9	56.1
512	896	2^{128}	2^{157}	528	384 kB	49.4	54.4
192	960	2^{192}	2^{276}	603	144 kB	26.1	103.0
384	1152	2^{192}	2^{249}	727	288 kB	34.8	77.2
256	1280	2^{256}	2^{368}	839	192 kB	21.3	126.4
512	1536	2^{256}	2^{332}	1013	384 kB	27.5	97.9

Table 6.3: Smoother VSH Variants

k	n	Coll	Pre	Memory	MB/s	Cycles/b
128	640	2^{128}	2^{184}	96 kB	66.9	40.2
256	768	2^{128}	2^{166}	192 kB	94.0	28.6
512	896	2^{128}	2^{157}	384 kB	97.9	27.5
192	960	2^{192}	2^{276}	144 kB	51.0	52.8
384	1152	2^{192}	2^{249}	288 kB	68.6	39.2
256	1280	2^{256}	2^{368}	192 kB	40.8	65.9
512	1536	2^{256}	2^{332}	384 kB	54.9	49.0

per byte and the speed of SHA-512 is 193 MB/s or 13.9 cycles per byte. Both were measured by the Crypto++ benchmark. Our fastest variant of Smoother VSH with 128-bit collision resistance approaches two thirds of the speed of SHA-256.

None of the seven variants of Faster VSH would be considered sufficiently secure based on the original computational VSSR assumption, possibly with the exception of the last one that is as secure as a 1013-bit RSA modulus.

6.4 Separation of Colliding Inputs

The list elements used in VSH are independent in a very strong sense that results in non-trivial separation of colliding messages. Suppose there is a pair of colliding inputs $m \neq m'$ under Faster or Smoother VSH such that

$$H(m) = \prod_{i=1}^{k} s_i \mod M \tag{6.12}$$

$$H(m') = \prod_{i=1}^{k} t_i \mod M \tag{6.13}$$

and the two inputs differ in precisely $l \leq k$ of the b-bit message blocks that select a particular element from a list. Given $H(m) = H(m')$, there is a congruence of two products of primes:

$$\prod_{i=1}^{l} s'_i = \prod_{i=1}^{l} t'_i \mod M \ . \tag{6.14}$$

At least one of the products must exceed M. This argument is analogous to that used by Tillich and Zémor to establish a similar property for their hash function. If d is the maximal bit length of any prime in our k lists, then

$$l \geq \frac{n}{d} \ .$$

More precisely, if the first $k2^b$ primes are used to fill the lists, the largest prime is approximately equal to $k2^b \log(k2^b)$. The length of that prime in bits is then

$$d \approx b + \log_2(bk \log 2 + k \log k) \ .$$

Any pair of colliding messages must differ in δ of the b-bit blocks where

$$\delta \approx \frac{n}{b + \log_2(bk \log 2 + k \log k)} \ .$$

Table 6.4: Minimum Distance of Collisions in New VSH Variants

k	n	δ
128	640	35
256	768	40
512	896	44
192	960	51
384	1152	58
256	1280	66
512	1536	75

A more exact value of δ can easily be computed as the least integer j such that the product $\prod_{i=0}^{j} p_{(k-i)2^b}$ exceeds M. The minimum separation constants for our seven variants are listed in Table 6.4. Observe that $\delta \approx n/20$ for all the cases considered.

As an example, take our most efficient variant with $n = 896$ and $k = 512$ and the separation constant equal to 44. This does not prevent colliding messages that differ in as few as 44 bits, but the differences cannot be limited to fewer than 44 blocks. Colliding messages are therefore easy to tell apart.

On the other end, if we select any 43 of the (aligned) 8-bit blocks and hash all the $2^{43\times 8} = 2^{344}$ messages obtained by varying these blocks in all the possible ways, then the resulting digests will all be different. All this for a function with estimated 2^{128} collision resistance.

Note that our collision separation property is valid for the compression function only and does not extend to the hash function operating in the Merkle-Damgård mode. Once the length of inputs is not restricted, there may exist colliding inputs that differ in as little as a single bit.

We do not have a more direct link to collision resistance, the property itself does not prevent collision attacks. After all, non-trivial separation of colliding inputs holds for the function of Zémor and Tillich as well as the LPS function where collisions are now easy to find.

6.5 Summary

The main advantage of Fast VSH was the heuristic connection to well-known hard algorithmic problems. In Faster VSH modulo an integer that is hard to factor, there is still the option to rely on the VSSR assumption. Our modification preserves the provable connection and even allows a conceptually simpler version of the security proof, because one does not need to deal with chaining at all. The design of Faster VSH also prevents the known attacks on Fast VSH. This does come at a cost that can be controlled.

Most importantly, the modifications proposed in this chapter allow the security to be measured in a different way. There is a deep combinatorial problem behind Faster VSH. The problem is very similar to problems behind the functions SWIFFT or FSB. The similarity allows us to adapt the attack methods and obtain meaningful bounds of security. With the new view of security, we no longer rely on the VSSR or VSDL assumption. This allows us to introduce smooth moduli in Section 6.2 and rely exclusively on the hardness of k-products.

The VSSR assumption can heuristically be linked to factoring and comes with a reasonable lower bound. On the other hand, VSDL is not that closely connected to discrete logarithms modulo M, it is an ad-hoc assumption that has never been quantified.

The new hard problem we propose can also be considered ad-hoc, but we believe it captures the security properties *better*. In addition, it comes with a quantified estimate of hardness. The estimates are type **lU** bounds at the moment and may not provide the most useful information about security as such. The bounds only quantify the cost of the tree algorithm. If we are confident that this is the most efficient strategy, we will be able to assume the type of the bounds is **L**. No hardness assumption in cryptology was created "standard". Although more insight in the complexity of these specific multiplicative k-sum problems is desirable, there is considerable evidence that k-sums are hard in at least some groups.

The most prominent feature of our new compression functions is the now reasonable performance. Smoother VSH in particular can compete with some of the "classical" hash functions widely deployed today. Further performance improvements may be possible.

Chapter 7

Field Smooth Hash

The major drawback of the functions from the previous chapter is the need to maintain a relatively big table of prime numbers. The table can either be hardcoded in the implementation, or generated at runtime. Both approaches affect set-up time as well as runtime performance. The few hundred kilobytes of memory required may be prohibitive for some architectures, in particular in hardware. Even if memory is cheap and fast, the size of tables of prime numbers significantly limits the freedom to select parameters. Values of b around 20 are highly impractical, values above 30 become impossible.

This chapter presents a new compression function family based on multiplication in finite fields $\mathbb{F}_{q^d}$ for $d > 1$. The main advantage of the function is that there is no more need to keep any table of the multiplicative weights. Our new design can generate the elements on the fly. The block size b can therefore be increased considerably with a positive effect on performance. Apart from the omnipresent connection to the k-sum problem, the new function also provides good collision separation. The minimum distance of collisions δ becomes a parameter that can be set more freely than before.

To motivate our design, we will briefly mention two related public-key cryptosystems based on knapsacks.

7.1 Chor-Rivest Cryptosystem

In a knapsack public key cryptosystems one typically publishes a set of weights $A = \{a_1, \ldots, a_q\}$. Messages are encrypted by encoding them as q-bit vectors and evaluating the knapsack function $E(m) = \sum_{i=1}^{q} m_i a_i$. The set A has a special secret structure that allows the legitimate recipient to invert the knapsack. The public-key system designed by Chor and Rivest is an additive knapsack scheme connected to arithmetic in finite fields [38, 37].

System Generation Let $\mathbb{F}_q$ be a finite field with q elements, pick an extension degree d such that discrete logarithms in $\mathbb{F}_{q^d}$ are feasible. Let $f(t) \in \mathbb{F}_q[t]$ be a monic degree d polynomial irreducible over $\mathbb{F}_q$, represent $\mathbb{F}_{q^d}$ as $\mathbb{F}_q[t]/(f(t))$. Select a random multiplicative generator g of $\mathbb{F}_{q^d}$. Fix a numbering α_i for $1 \leq i \leq q$ of the field $\mathbb{F}_q$ and compute the logarithms

$$\gamma_i = \log_g(t + \alpha_i) \ . \tag{7.1}$$

Select a random permutation $\pi : \{1, \ldots, q\} \to \{1, \ldots, q\}$ and a random constant $0 \leq u \leq q^d - 2$. Compute $a_i = \gamma_{\pi(i)} + u$ for $i = 1, \ldots, q$. The public key consists of $A = \{a_1, \ldots, a_q\}$ as well as d and q. The mapping π and the elements t, u are in the private key.

From a theorem by Bose and Chowla it follows that all the possible sums of precisely d of the elements γ_i are distinct modulo $q^d - 1$ [26]. To prove the theorem, let $1 \leq i_1 \leq \ldots \leq i_d \leq q$ and $1 \leq j_1 \leq \ldots \leq j_d \leq q$ be two sets of indices such that the sums $\gamma_{i_1} + \gamma_{i_2} + \ldots + \gamma_{i_d}$ and $\gamma_{j_1} + \gamma_{j_2} + \ldots + \gamma_{j_d}$ are equal. The following powers of g are then also equal in $\mathbb{F}_{q^d}$:

$$g^{\gamma_{i_1}} \times \ldots \times g^{\gamma_{i_d}} = g^{\gamma_{j_1}} \times \ldots \times g^{\gamma_{j_d}} \ . \tag{7.2}$$

By the definition (7.1)

$$\prod_{k=1}^{d} (t + \alpha_{i_k}) = \prod_{k=1}^{d} (t + \alpha_{j_k}) \ . \tag{7.3}$$

If the term t^d is subtracted from both sides, we obtain an equation of two polynomials of degree at most $d - 1$ in t over $\mathbb{F}_q$. Because t has algebraic degree d, it follows that $i_k = j_k$ for $1 \leq k \leq d$.

Encryption Assume that the message m is encoded as an q-bit binary vector $(m_1, \ldots, m_q)$ with weight precisely d. Up to $\binom{q}{d}$ bit messages can be encoded in this form. With an appropriate encoder the length of the plaintexts is $\lfloor \log_2 \binom{q}{d} \rfloor$. To encrypt m, compute

$$E(m) = \sum_{i=1}^{q} m_i a_i \mod q^d - 1 \tag{7.4}$$

Encryption only involves modular additions and results in an integer in the range $0, \ldots, q^d - 1$.

Decryption To decrypt $s = E(m)$, first compute $s' = s - du \mod p^d - 1$. Then obtain the extension field element $q(t) = g^{s'}$ represented as a degree $d-1$ polynomial in t over $\mathbb{F}_q$. Then the polynomial $f(t) + q(t)$ factors in $\mathbb{F}_q[t]$ to linear terms

$$f(t) + q(t) = (t + \alpha_{i_1}) \cdot (t + \alpha_{i_2}) \cdots (t + \alpha_{i_d}) \ . \tag{7.5}$$

Find the roots by successive substitutions of elements from $\mathbb{F}_q$. The d positions of 1 bits in m can then be recovered by applying π^{-1} to the indices $i_1, \ldots, i_d$.

Proposed Parameters Key generation involves computations of discrete logarithms in $\mathbb{F}_{q^d}$. The parameters need to be selected such that the logarithms are feasible. It is suggested that d is a smooth number to allow fast DL computations by the algorithm of Pohlig and Hellman. One of the parameters sets considered by Chor and Rivest is $q = 197$ and $d = 24$. The encryption function then maps $\log_2 \binom{197}{24} \approx 101$ bits to $d \log_2 q \approx 183$ bits.

7.1.1 Cryptanalysis

The density of the underlying additive knapsacks is greater than one, for example in the case $q = 197$ and $d = 24$ it is approximately 1.077. The cryptosystem has resisted lattice attacks for several years. Schnorr and Hörner demonstrated that random instances with parameters $q = 103, d = 12$ and $q = 151, d = 16$ were feasible with an improved lattice reduction algorithm [126].

Vaudenay exploited the smoothness requirement on d in a practical algebraic attack that efficiently recovers the secret key from the public key for virtually all reasonable parameter sets [143, 144]. If d has a factor r greater than $\sqrt{d + \frac{1}{4}} + \frac{1}{2}$, the complexity of the attack is $O(d^3 q^r / r^2)$. An instance of the cryptosystem with both q and d prime would survive the attack. Such parameters however limit the ability to compute discrete logarithms necessary for key generation.

7.2 Powerline System

Lenstra designed a similar public-key system that does not involve computation of discrete logarithms to generate the parameters [80]. Operations are performed directly in the multiplicative group of the finite field $\mathbb{F}_{q^d}$.

System Generation Let q be a power of a prime, let $f(x) \in \mathbb{F}_q[x]$ be an irreducible polynomial of degree d. The field $\mathbb{F}_{q^d}$ will be publicly represented as $\mathbb{F}_q[x]/(f(x))$. Select an integer $k \leq q$ and a random injective mapping $\pi : \{1, 2, \ldots, k\} \to \mathbb{F}_q$. Pick a random element $t \in \mathbb{F}_{q^d}$ with algebraic degree d over $\mathbb{F}_q$ such that $\mathbb{F}_{q^d} = \mathbb{F}_q(t)$. Fix a random $u \in \mathbb{F}_{q^d}$ and a random integer v satisfying $1 \leq v \leq q^d - 1, \;\; \gcd(v, q^d - 1) = 1$. For each $1 \leq i \leq k$ compute the element $a_i = u^v (t - \pi(i))^v$ of $\mathbb{F}_{q^d}$.

Publish a description of the field $\mathbb{F}_q$ and the polynomial $f(x)$ defining the extension $\mathbb{F}_{q^d}$. The public key consists of k and the elements $a_1, a_2, \ldots a_k$. Keep the t, u, v and the function π secret.

Encryption A message is assumed to be represented by sequence of non-negative integers $m = (m_1, \ldots, m_k)$ such that $\sum_{i=1}^k m_i = d$. To encrypt m, compute the element $E(m) = \prod_{i=1}^k a_i^{m_i}$ in $\mathbb{F}_{q^d}$. The encryption function is a multiplicative knapsack.

Decryption Compute $z = v^{-1} \mod q^d - 1$, the multiplicative inverse of v and the element u^{-d} of $\mathbb{F}_{q^d}$. To decrypt $E(m)$, compute $E(m)^z \cdot u^{-d} - t^d$ and express it in the basis $1, t, \ldots, t^{d-1}$ of $\mathbb{F}_{q^d}$:

$$E(m)^z \cdot u^{-d} - t^d = \sum_{i=0}^{d-1} c_i t_i, \quad c_i \in \mathbb{F}_q \; .$$

The message m can then be recovered by finding the roots of the polynomial

$$t^d + \sum_{i=0}^{d-1} c_i t^i = 0$$

over $\mathbb{F}_q[t]$. For every $1 \leq i \leq k$, the number m_i will be the multiplicity of $\phi(i)$ as a zero of the above polynomial.

Security The Powerline system is at least as secure as the Chor-Rivest system. In particular, instances of the latter system can be transformed to instances of the Powerline system. For a generator $g \in \mathbb{F}_{q^d}$, if a_i are the weights of the additive Chor-Rivest Knapsack, then the powers g^{a_i} define a multiplicative Powerline knapsack.

Lenstra also suggests that the fastest method to break the multiplicative Powerline system is to transform it to an instance of the Chor-Rivest system by taking discrete logarithms and attack the additive knapsack instead. By the results of Vaudenay, it is desirable that both q and h be prime even in the multiplicative case of the Powerline system. In such a case, the fastest method to solve the knapsack in the additive representation is due to Brickell [38].

7.3 A New Compression Function

Our search for group-based hash functions within this thesis has become a search for hard group knapsacks with compact representation. We design a knapsack and a compression function inspired by the two above cryptosystems.

For hashing purposes, the knapsack function should not be injective. If we drop the requirement that the weight of the input message m is precisely d, we obtain a compression function. By the result of Bose and Chowla, any pair of inputs $m \neq m'$ that collide under the Chor-Rivest knapsack must differ in at least $\delta = d + 1$ bits, an equivalent result is valid for the Powerline system. This is the collision separation property we obtained for our functions in the previous chapter.

The fastest known method to break Powerline system seems to attack it in the additive representation after computing discrete logarithms. The hardness of the lattice methods involved depends on the density and the dimension represented by the total number of weights. One may therefore simply generate a random high density Powerline knapsack with large k and then discard the private key. The weights a_i in the powerline system unfortunately do not have a compact representation, they are in general full length elements of the extension field. In terms of storage such a knapsack is no better than a list of k random elements of $\mathbb{F}_{q^d}$. If we include the private key in the description of the powerline knapsack, it will be possible to generate the weights a_i from the very compact description involving t, u, v and π. If all the parameters are public, one may as well choose $u, v = 1$ and $t = x$.

The First Knapsack Let $\mathbb{F}_q$ be a field with q elements and let the extension $\mathbb{F}_{q^d}$ be defined with respect to a degree d monic irreducible polynomial $f(t) \in \mathbb{F}_q[t]$. Fix a positive integer $k \leq q$ and an injective mapping $\pi : \{1, \ldots, k\} \to \mathbb{F}_q$. Consider the multiplicative knapsack function H defined on the elements $a_i = (t + \alpha_i)$ such that $\alpha_i = \pi(i)$ for $i \in 1, \ldots, k$. It transforms a k-bit vector m to the product

$$H(m) = \prod_{i=1}^{k} {a_i}^{m_i} \tag{7.6}$$

computed in $\mathbb{F}_{q^d}$. If $k > d \log_2 q$, this function compresses the input. It is an instance of group subset sums considered by Impagliazzo and Naor (cf. Section 4.4) for G the multiplicative group of $\mathbb{F}_{q^d}$.

The above knapsack serves as an intermediate step towards a more practical construction. We therefore propose no specific parameter choices and, in particular, make no claims about preimage or collision resistance of H. The function may turn out to be easy to invert for poor parameter choices.

By construction, any pair of colliding inputs to H differ in at least $\delta = d+1$ bits. Any of the k weights a_i can be generated on the fly given suitable π. Evaluation of the H requires a single multiplication in $\mathbb{F}_{q^d}$ per input bit. One of the operands is linear in t, the other is in general full-length in $\mathbb{F}_{q^d}$.

The expression (7.6) is analogous to the product of primes $\prod_{i=1}^{k} p_i^{m_{jk+i}}$ mod M that appears in Basic VSH (cf. p. 57). Because storage of the multiplicative weights a_i is now free, we may increase the number of knapsack elements to $k2^b$ and perform one group operation per b bits for a constant $b > 1$. This is the same "standard" trick that led to Fast VSH and was in fact used in most of the functions in Chapter 5.

Field Smooth Hash Let $\mathbb{F}_{q^d}$ be a finite field as above, let G denote the multiplicative group of $\mathbb{F}_{q^d}$ and let $n \approx \log_2 |G|$. Instead of a single mapping π consider a set of k injective functions $\phi_i : \{0,1\}^b \to L_i$ for $i = 1, \ldots, k$ and L_i pairwise disjoint such that $L_i \subseteq \{(t+\alpha) \in \mathbb{F}_{q^d} \mid \alpha \in \mathbb{F}_q\}$ and $|L_i| = 2^b$. The above condition implies $k2^b < q$, in order for our function to compress, require also $bk > n$. If m is a bk-bit input message, compute the compression function as follows:

1. Initialize $x_0 = 1$.
2. For $i = 1$ to k compute $x_{i+1} = x_i \times \phi_i(m[i])$.
3. Return x_{k+1}.

The result is an element of the field $\mathbb{F}_{q^d}$ that can be represented by approximately n bits. The function can be seen as a variant of Faster VSH. Small primes are replaced by low degree polynomials, reduction modulo the integer M is replaced by reduction modulo the degree d polynomial $f(t)$.

7.4 Security

The construction leaves a lot of freedom for selecting parameters. Because Step 2 is executed once per b bits of input, one would want b to be as big as possible to achieve the highest performance. Unfortunately, if b is selected to be the largest integer subject to the condition $k2^b < q$, the compression function may turn out to be easy to invert.

Factoring Attacks

If we omit reductions modulo $f(t)$ from FSH, the function maps the bk bits of input to a degree k monic polynomial $x'(t) \in \mathbb{F}_q[t]$. This mapping is injective and easy to invert by factoring the polynomial $x'(t)$ over $\mathbb{F}_q$. There exist efficient probabilistic methods for doing so, for example the algorithm by Cantor and Zassenhaus [34].

Given a digest $x \in \mathbb{F}_{q^d}$, one may invert FSH by finding a suitable monic polynomial $z(t) \in \mathbb{F}_q[t]$ such that $x'(t) = x + z(t)f(t)$ factors over $\mathbb{F}_q$:

$$x'(t) = \prod_{i=1}^{k} s_i \quad \text{for } s_i \in L_i \ . \tag{7.7}$$

The degree of $z(t)$ equals $k - d$, there are q^{k-d} such polynomials available. For a random $z(t)$, there is a chance $2^{k(b-\log_2 q)}$ that the factorization of $x'(t)$ has the form (7.7) required to invert H. One expects to succeed after $2^{k(\log_2 q-b)}$ trials. Inversion by factoring gets easier as b approaches $\log_2 q$. On the other hand, the approach can be slower than generic preimage search for many parameter choices.

Divide and Factor To extend the above inversion attack, one can guess $1 \le j \le k-d$ message blocks $m[i]$ and solve for the remaining $k-j$. Without loss of generality fix the blocks numbered 1 to j and invert the function on positions $j+1, \ldots, k$. Let $s_i \in L_i$ for $i = 1, \ldots, j$ be the linear polynomials corresponding to the bj bits guessed. To invert $x \in \mathbb{F}_{q^d}$, first compute

$$y = x \times \left(\prod_{i=1}^{j} s_i \right)^{-1} \in \mathbb{F}_{q^d} \ . \tag{7.8}$$

Then find a degree $k - d - j$ polynomial $z(t)$ such that $y'(t) = y + z(t)f(t)$ factors over $\mathbb{F}_q$ as $y'(t) = \prod_{i=j+1}^{k} s_i$ for $s_i \in L_i$. There are 2^{bj} choices for

$s_1, \ldots, s_j$ and q^{k-d-j} possible values of $z(t)$. The expected number of trials needed to invert H is $2^{(k-j)(\log_2 q - b)}$. The expression is minimized when $j = k - d$ and evaluates to 2^{n-bd}. In this case the degree of $z(t)$ is zero, i.e. $z(t) = 1$ and $y'(t) = y + f(t)$.

If we fix $j > k - d$ of the k available positions, then the function can be inverted by computing $y \in \mathbb{F}_{q^d}$ as in (7.8) and then factoring it in $\mathbb{F}_q[t]$. The degree of y is at most $d - 1$. The probability that it factors as $y = \prod_{i=j+i}^{k} s_i$ is $2^{b(k-j)-n}$, therefore $2^{n-b(k-j)}$ trials are expected to succesfully invert H. This cost is strictly greater than the optimal value 2^{n-bd} achieved for $j = k - d$. Observe that when $j = k$, we are guessing all the bk bits of the input and expect to succeed in 2^n trials.

Countermeasures To prevent the factoring attacks, we need to select the parameters such that the cost 2^{n-bd} is large. As $k2^b$ approaches the maximum $q = 2^{n/d}$, the quantity 2^{n-bd} approaches k^d. Factoring attacks are a threat in particular for instances with small k and d. We will nevertheless keep track of $n - bd$ when selecting parameters.

Extended k-Tree Algorithm

If $n - bd$ is sufficiently high to prevent the above inversion attacks, measure security by the cost of the extended tree algorithm as in the case of Faster VSH in Chapter 6. For some fields, the tree algorithm may run in a chain of multiplicative subgroups of G. In other fields, it may be necessary to compute discrete logarithms first and proceed in the equivalent additive representation of G. As in the case of Faster VSH, the cost of discrete logarithms will be neglected. The type **IU** bounds remain valid. Recall that we will quantify preimage resistance with respect to the tree algorithm as $2^{\frac{n}{v}}$ for v that satisfies $\frac{2^{v-1}}{v} = r = \frac{bk}{n}$, collision resistance will be approximated by $2^{\frac{n}{v'}}$ where v' satisfies $\frac{2^{v'-1}}{v'} = 2r$.

7.5 Choosing the Base Field

Apart from the above security considerations, the choice of parameters is restricted by the existence of a field for given q and d. The limitations are fortunately not too severe, there are a plenty of fields to choose from. We can even be quite picky about the fields to hope for better performance.

We will choose the field $\mathbb{F}_q$ such that its elements (almost) fill an integral number of machine words, and stick to 64-bit architecture. If we denote the

Table 7.1: Available Extensions of Binary Fields

Field Size	Degrees of Available Trinomials
2^{63}	$2, 4, 5, 10, 11, 17, 20, 22, 23, 25, 29, 31$
2^{64}	$3, 5, 7, 9, 11, 15, 17, 21, 23, 25, 29, 31$
2^{127}	$3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15$
2^{128}	$3, 5, 7, 9, 11, 15$
2^{191}	$3, 4, 5, 6, 7, 9, 10$
2^{192}	$5, 7$
2^{255}	$2, 4, 7$
2^{256}	$3, 5, 7$

number of words needed to represent the base field $\mathbb{F}_q$ by w, then $\log_2 q \approx 64w$. The bitlength of the extension field is then $n \approx 64dw$. To define the extension field $\mathbb{F}_{q^d}$, we only consider irreducible monic trinomials $f(t) = t^d \pm t^c \pm 1$ for $1 < c < d$. This way we will avoid field multiplications when reducing modulo $f(t)$ and accomplish such reductions only by a few additions or subtractions in $\mathbb{F}_q$. The candidate polynomials can be easily tested for irreducibility, we found all the useful pairs of q and the corresponding $f(t)$ by a simple program for Sage [134].

Binary Fields In the binary case, the choice of trinomials f is limited, for a given d there are only $d - 1$ candidate polynomials of the form $f(t) = t^d + t^c + 1$. Moreover, if the extension degree of $\mathbb{F}_q$ over $\mathbb{F}_2$ is even, then all the trinomials of this form with d even will be reducible over $\mathbb{F}_q$. For this reason we will consider either $q = 2^{64w}$ or $q = 2^{64w-1}$ for the base field. We tested such fields for $w \in \{1, 2, 3, 4\}$ for all d such that $dw \leq 32$. The latter condition simply avoids the fields that lead to $n > 2048$. The available choices for d are listed in Table 7.1.

Prime Fields We will consider fields for q an odd prime number only slightly below 2^{64w}. Because $1 \neq -1$ in $\mathbb{F}_q$, there are now $4(d-1)$ candidate polynomials to test for irreducibility. For any chosen $w \geq 1$ and $d \geq 2$ we were able to find a valid pair $q, f(t)$ practically in no time, by trying random q close to 2^{64w}.

To optimize computation in $\mathbb{F}_q$ we also considered special prime moduli

Table 7.2: Available Extensions of Fields Modulo Special Primes

Field Size	Degrees of Available Trinomials
$2^{127} - 1$	2,4,5,8,9,10,11,12,13,16
$2^{127} - 735$	2,4,6,8,9,13
$2^{255} - 19$	4,5,7,9

q that allow cheap reductions. The available extension degrees for three such primes are displayed in Table 7.2.

Even if we aim for fields that can be represented and implemented efficiently, there is still plenty of freedom for selecting parameters. We continue to look for convenient parameters experimentally.

7.6 Implementation

For simplicity, only consider multiples of 8 for b. We may then process messages in blocks of B bytes for $B = b/8$.

7.6.1 From Bits to Field Elements

We proceed to define the k injective mappings $\phi_i : \{0,1\} \to L_i \subseteq \mathbb{F}_q$ that map message blocks $m[i]$ to elements of the base field. The index i can be represented by $\log_2 k$ bits. Take the binary representation of i and append the b-bit string input to ϕ_i, i.e. concatenate the index i and the b-bit message block $m[i]$ to form $\alpha = i||m[i]$. Because $k2^b < q$ implies $\log_2 k + b < \log_2 q$, the resulting bit string is at most $\log_2 q$ bits long. Pad α to fill w machine words such that the result is a binary representation of an element of $\mathbb{F}_q$. We go for the simplest padding with all zero bits, in practice one may choose any fixed pattern. The total size of the padding pattern across all $1 \leq i \leq k$ is approximately $k(\log_2 q - \log_2 k - b)$ bits. The pattern can be considered extra "key material" that allows to select a custom compression function even if all the remaining parameters are fixed. Define the output of $\phi_i(m[i])$ as the element of $\mathbb{F}_q$ with binary representation α.

In practice, we will simply represent the index i by an integral number of bytes, and append B bytes of input. The function ϕ_i can be "computed" for free, it does nothing more than read the input. This special choice of ϕ_i removes the requirement to store the lists of field elements.

7.6.2 The FSH Iteration

By the similarity to Faster VSH, the new function allows an equivalent iterative implementation:

1. Initialize a $w(d+1)$-word intermediate value x with $\phi_1(m[1])$.
2. For $i = 2$ to $d-1$ multiply x by $(t + \phi_i(m[i]))$, no reductions by $f(t)$ are needed here.
3. For $i = d$ to k multiply x by $(t + \phi_i(m[i]))$ modulo $f(t)$.
4. Output x.

A multiplication of x by $(t + \alpha)$ can be implemented by adding the vector x shifted w words to the left and the product $x\alpha$. Once x has full length of d field elements, the product in Step 3 needs d field multiplications and $d-1$ additions. The resulting polynomial is a degree $d+1$ polynomial over $\mathbb{F}_q$. A reduction modulo $f(t)$ can be accomplished by two additions resp. subtractions in $\mathbb{F}_q$.

7.6.3 Field Arithmetic

We have implemented FSH in C for a 64-bit PC architecture for both binary and prime extension fields. The initial implementations were for $w = 1$ and allowed the values B, k, d to be specified at compile time. A later version allows to compile for various w as well. With such a versatile implementation, we may not have reached the performance that would be expected of a fine-tuned optimized implementation specific to a single set of parameters. The advantage of the approach is that real-world performance can help select parameters from the vast set of possibilities.

Binary Fields

Field addition in binary fields can be implemented by XORs. We used routines from the MPFQ library by Gaudry and Thomé for multiplication [54]. The library offers optimized arithmetic routines for many finite fields, it is especially suitable in our setting when a field is known at compile time. For the fields we consider, all the MPFQ code is actually inlined, i.e. there is no binary library to call.

A field multiplication for $w = 1$ performs a 64×64 bit carry-less binary multiplication. Intel recently extended the x86-64 instruction set by the instruction `PCLMULQDQ` to compute such a product [60]. We implemented an option to call the instruction instead of the MPFQ multiplication routine where available.

For $w \geq 2$ we exclusively rely on MPFQ for both multiplication and addition, the library provides all binary fields up to 2^{256}, this limits us to $w \in \{1, 2, 3, 4\}$.

For all the binary fields, we use the fields as implemented in MPFQ. Field operations in $\mathbb{F}_q$ are the most basic arithmetic primitives we use, we will therefore not describe inner workings of $\mathbb{F}_q$. In order to implement the mapping ϕ_i properly, it is sufficient to check that the field with 2^{64w} elements is represented in MPFQ by an array of w machine words.

Prime Fields

An implementation for $w = 1$ was written from scratch in C with limited inline assembly code for modular arithmetic. Divisions are avoided by performing Montgomery reductions instead. There is no need to perform any conversions to Montgomery representation, the reduction is linear over $\mathbb{F}_q$ in the end it only permutes the results. As in the case of Faster VSH, the final output can be "fixed", but it is not necessary.

For $w \geq 2$ we use the MPFQ library for all arithmetic in prime order fields. The library provides optimized routines for three special moduli numbers that allow fast reductions, these are the primes listed in Table 7.2. There are dedicated routines for general primes q known to fit to w machine words, the arithmetic is in turn performed by the GNU MP library. The library offers routines to compute in Montgomery representation, these are used directly, any conversions are omitted.

7.7 Experimental results

Thanks to the versatile implementation we were able to compile many instances and perform preliminary speed tests. We report on a selection of variants including the most efficient ones found when testing on the same Intel Core i7 system used in Chapter 6. Speed was measured when hashing long messages in Merkle-Damgård mode.

The choice of parameters was guided by the complexity collision search by the extended tree algorithm. We set three target levels for this quantity, 2^{128}, 2^{192} and 2^{256}. The results are organized with respect to the class of $\mathbb{F}_q$ and ordered by estimated collision resistance.

7.7.1 Binary Fields

The parameters and performance results are listed in Table 7.3. For a given security level, the performance of the variants included increases with w, i.e. size of the base field $\mathbb{F}_q$. The fastest variant that is expected to offer an equivalent of 2^{128} collision resistance is a sum on $k = 11$ “lists” of $2^b = 2^{184}$ elements each. Such extreme values of b are possible because all the weights $(t + \alpha_i)$ are generated on the fly.

The code for $w = 4$ uses multiplication from the MPFQ library. The dedicated `PCLMULQDQ` instruction appears to be, perhaps surprisingly, slower than MPFQ. More careful low-level optimization could possibly benefit from the new instruction.

7.7.2 Prime Fields

The results for general primes q are listed in Table 7.4 and for the three special primes in Table 7.5. The software performance of FSH over prime fields is superior to the case of binary fields. Here as well the fastest variants are those with large w. The variants with $w = 1$ benefit from a more careful dedicated implementation, the instances for $w \geq 2$ use a more generic code and are likely to suffer from function call overhead. As soon as $w = 4$, even the generic code becomes faster for a given security level.

Observe in particular the function with $w = 7, k = 11, d = 2$ in row 4 of Table 7.4. It offers an equivalent of 2^{130} collision resistance and computes in a quadratic extension of a field modulo a 448 bit prime number. The hypothetical lists of weights are 2^{352} elements long, only 11 linear polynomials are multiplied to compress 3872 bits. This function runs at approximately half the speed of SHA-256.

The fastest compression function found corresponds to the second row of Table 7.5. It offers an equivalent of 2^{130} collision resistance. Computation is preformed over a base field modulo the special prime $q = 2^{255} - 19$. The function benefits from cheap reductions in the base field in addition to the usual cheap reductions in the extension field. This function runs at approximately 70% of the speed of SHA-256. This is better than the fastest variant of Smoother VSH (p. 77).

7.8 Summary

The new family of compression functions in multiplicative groups of finite fields is both simple and flexible in design. In contrast to VSH there is no

Table 7.3: Variants of FSH in Binary Fields

w	B	k	d	n	Pre	Coll	$n - bd$	MB/s	Cycles/b
1	6	120	15	960	2^{154}	2^{128}	240	22.6	119
2	12	21	6	768	2^{167}	2^{128}	190	28.7	94
4	23	11	3	768	2^{167}	2^{128}	216	38.3	70
1	5	161	22	1408	2^{232}	2^{192}	352	11.9	226
2	9	34	11	1152	2^{252}	2^{193}	360	17.1	157
4	24	25	5	1280	2^{240}	2^{193}	320	27.1	99
4	24	42	7	1792	2^{315}	2^{257}	448	20.1	134

Table 7.4: Variants of FSH in Prime Fields for General q

w	B	k	d	n	Pre	Coll	$n - bd$	MB/s	Cycles/b
1	6	85	14	896	2^{157}	2^{128}	224	46.1	58
2	12	42	7	896	2^{157}	2^{128}	224	40.1	67
4	25	40	4	1024	2^{152}	2^{128}	224	65.2	41
7	44	11	2	896	2^{160}	2^{130}	192	78.3	34
1	6	101	20	1280	2^{240}	2^{192}	320	29.5	91
2	12	50	10	1280	2^{240}	2^{192}	320	27.2	99
4	24	25	5	1280	2^{240}	2^{192}	320	45.9	59
1	6	120	26	1664	2^{322}	2^{256}	416	22.8	118
2	12	60	13	1664	2^{322}	2^{256}	416	20.8	129
4	24	42	7	1792	2^{315}	2^{257}	448	34.4	78

Table 7.5: Variants of FSH in Prime Fields for Special q

w	B	k	d	n	Pre	Coll	$n - bd$	MB/s	Cycles/b
2	12	85	8	1024	2^{151}	2^{128}	256	53.3	51
4	25	40	4	1024	2^{153}	2^{128}	224	111.9	24
2	12	50	10	1280	2^{240}	2^{192}	320	38.6	70
4	25	24	5	1280	2^{240}	2^{192}	320	77.9	35
2	12	60	13	1664	2^{322}	2^{256}	416	29.5	91
4	24	42	7	1792	2^{315}	2^{257}	448	60.1	45

need to collect and organize prime numbers. The fields in the new setting can be made to measure, there is enough freedom to satisfy the limitations of almost any architecture. The linear polynomials can be obtained and multiplied fast, memory requirements hardly exceed the output size. For well chosen parameters, the performance of Field Smooth Hash in software is superior to Faster and Smoother VSH considered in Chapter 6. The performance of our fastest variant that offers 128-bit collision resistance is comparable to SHA-256.

Security properties are again linked to the hardness of specific k-product problems in a multiplicative group. The security is quantified by means of type **lU** bounds relative to the cost of the tree algorithm.

Conclusions

The recent "provably secure" hash functions have turned out to be surprisingly similar. From a high level point of view, all the functions considered in Chapters 4 to 7 are k-sums in finite groups. This allows to establish a provable link between the security of such functions and the hardness of the corresponding k-sum instances. In this respect, the generalized birthday problem appears to be well suited for hash function design.

A connection to some mathematical problem does not of course *prove* security by itself. Several hash functions related to Cayley graphs were claimed to be provably secure and then shown insecure. The current version of FSB has evolved from a series of "provably secure" and broken variants. In these cases, security was provably reduced to a problem that turned out to be easy. Most of the attacks on k-sum compression functions either applied the tree algorithm or exploited some special structure of the group elements involved. The complexity of the tree algorithm can be factored in when designing new functions. The k-sum problem is by no means new. It has been studied for several years and can be considered for a hardness assumption on its own.

The risk associated with special structure might not be easy to avoid. Such structure is needed for a k-sum function to have a compact description and in turn allow efficient implementation.

Finding k-sum instances that lead to both efficient and secure compression functions remains an open question. Our contributions show that even multiplicative k-sums can be very fast. We have also investigated an interesting collision separation property that appears to be exclusive to multiplicative k-sums.

The security of our new functions, as far as provable **L** bounds are concerned, remains open.

Bibliography

[1] L. M. Adleman and M.-D. A. Huang. Function field sieve method for discrete logarithms over finite fields. *Inf. Comput.*, 151(1-2):5–16, 1999.

[2] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[3] M. Ajtai. The Shortest Vector Problem in l_2 is *NP*-hard for Randomized Reductions (Extended Abstract). In *STOC*, pages 10–19, 1998.

[4] M. Ajtai. Generating hard instances of lattice problems. J. Krajíček (ed.), Complexity of computations and proofs. Quaderni di Matematica 13, 1-32., 2004.

[5] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997.

[6] Y. Arbitman, G. Dogon, V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFTX: A Proposal for the SHA-3 Standard. Submission to NIST, 2008.

[7] D. Augot, M. Finiasz, P. Gaborit, S. Manuel, and N. Sendrier. SHA-3 proposal: FSB. Submission to NIST, 2008.

[8] D. Augot, M. Finiasz, and N. Sendrier. A Fast Provably Secure Cryptographic Hash Function. 2003.

[9] D. Augot, M. Finiasz, and N. Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In E. Dawson and S. Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer, 2005.

[10] J.-P. Aumasson and W. Meier. Analysis of multivariate hash functions. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2007.

[11] M. Bardet, J. Faugere, and B. Salvy. On the complexity of Grobner basis computation of semi-regular overdetermined algebraic equations. *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[12] P. S. L. M. Barreto and V. Rijmen. The Whirlpool hashing function. Submitted to NESSIE, September 2000. Revised May 2003. Available: `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`.

[13] R. Barua and T. Lange, editors. *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, volume 4329 of *Lecture Notes in Computer Science*. Springer, 2006.

[14] A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.

[15] M. Bellare. Practice-oriented provable security. In I. Damgård, editor, *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1998.

[16] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2004.

[17] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997.

[18] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[19] C. Berbain, H. Gilbert, and J. Patarin. QUAD: A practical stream cipher with provable security. In Vaudenay [145], pages 109–128.

[20] E. R. Berlekamp, R. J. McEliece, and H. C. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. Inf. Theory*, 24:384–386, 1978.

[21] D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. FSBday. In Roy and Sendrier [119], pages 18–38.

[22] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Really fast syndrome-based hashing. In A. Nitaj and D. Pointcheval, editors, *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2011.

[23] E. Biham and O. Dunkelman. A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`.

[24] O. Billet, M. J. B. Robshaw, and T. Peyrin. On building hash functions from multivariate quadratic equations. In J. Pieprzyk, H. Ghodosi, and E. Dawson, editors, *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2007.

[25] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Yung [151], pages 320–335.

[26] R. C. Bose and S. Chowla. Theorems in the additive theory of numbers. *Comment. Math. Helv*, pages 141–147, 1962.

[27] J. Bosset. Contre les risques d'altération, un systeme de certification des informations. *01 Informatique*, 107, 1977.

[28] 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.

[29] J. Buchmann and R. Lindner. Secure parameters for SWIFFT. In Roy and Sendrier [119], pages 1–17.

[30] P. Camion. Can a fast signature scheme without secret key be secure? In A. Poli, editor, *AAECC*, volume 228 of *Lecture Notes in Computer Science*, pages 215–241. Springer, 1984.

[31] P. Camion and J. Patarin. The knapsack hash function proposed at Crypto'89 can be broken. In *EUROCRYPT*, pages 39–53, 1991.

[32] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[33] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Inf. Theory*, 44(1):367–378, 1998.

[34] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comput.*, 36:587–592, 1981.

[35] D. X. Charles, K. E. Lauter, and E. Z. Goren. Cryptographic hash functions from expander graphs. *J. Cryptology*, 22(1):93–113, 2009.

[36] C. Charnes and J. Pieprzyk. Attacking the SL_2 hashing scheme. In J. Pieprzyk and R. Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 322–330. Springer, 1994.

[37] B. Chor and R. L. Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. In *CRYPTO*, pages 54–65, 1984.

[38] B. Chor and R. L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

[39] H. Cohen, G. Frey, R. M. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* Discrete Mathematics and its Applications. Boca Raton, FL: Chapman & Hall/CRC. xxxiv, 808 p., 2006.

[40] S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In Vaudenay [145], pages 165–182.

[41] J.-S. Coron and A. Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013, 2004.

[42] 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.

[43] R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[44] I. Damgård. A design principle for hash functions. In Brassard [28], pages 416–427.

[45] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22:644–654, 1976.

[46] M. Finiasz. Syndrome based collision resistant hashing. In J. Buchmann and J. Ding, editors, *PQCrypto 2008*, volume 5299 of *LNCS*, pages 137–147. Springer, 2008.

[47] M. Finiasz, P. Gaborit, and N. Sendrier. Improved fast syndrome based cryptographic hash functions. *ECRYPT Hash Function Workshop 2007*, 2007.

[48] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Asiacrypt 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, 2009.

[49] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009.

[50] P.-A. Fouque and G. Leurent. Cryptanalysis of a Hash Function Based on Quasi-cyclic Codes. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2008.

[51] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In Smart [132], pages 31–51.

[52] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[53] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[54] P. Gaudry and E. Thomé. *MPFQ Library Version 1.0-rc3*, 2010. `http://www.mpfq.org`.

[55] P. Godlewski and P. Camion. Manipulations and errors, detection and localization. In *EUROCRYPT*, pages 97–106, 1988.

[56] O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42), 1996.

[57] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In J. Burton S. Kaliski, editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

[58] D. M. Gordon. Discrete logarithms in $GF(p)$ using the number field sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.

[59] M. Grassl, I. Ilic, S. S. Magliveras, and R. Steinwandt. Cryptanalysis of the Tillich-Zémor hash function. *J. Cryptology*, 24(1):148–156, 2011.

[60] S. Gueron and M. E. Kounavis. Intel carry-less multiplication instruction and its usage for computing the GCM mode - rev 2, 2010. Intel Software Network.

[61] S. Halevi, editor. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*. Springer, 2009.

[62] N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.

[63] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *FOCS*, pages 236–241. IEEE, 1989.

[64] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.

[65] A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[66] A. Joux and L. Granboulan. A practical attack against knapsack based hash functions (extended abstract). In *EUROCRYPT*, pages 58–66, 1994.

[67] A. Joux and R. Lercier. The function field sieve in the medium prime case. In Vaudenay [145], pages 254–270.

[68] A. Joux, R. Lercier, N. P. Smart, and F. Vercauteren. The number field sieve in the medium prime case. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 326–344. Springer, 2006.

[69] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *J. Cryptology*, 11(3):161–185, 1998.

[70] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[71] J. Kelsey and T. Kohno. Herding hash functions and the nostradamus attack. In Vaudenay [145], pages 183–200.

[72] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [43], pages 474–490.

[73] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. J. J. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.

[74] N. Koblitz and A. Menezes. Another Look at “Provable Security”. II. In Barua and Lange [13], pages 148–175.

[75] N. Koblitz and A. Menezes. Another look at “provable security”. *J. Cryptology*, 20(1):3–37, 2007.

[76] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[77] A. K. Lenstra and H. W. Lenstra. *The Development of the Number Field Sieve.* Springer, 1993.

[78] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[79] A. K. Lenstra, D. Page, and M. Stam. Discrete logarithm variants of VSH. In Nguyen [102], pages 229–242.

[80] H. W. Lenstra. On the Chor-Rivest knapsack cryptosystem. *J. Cryptology*, 3(3):149–155, 1991.

[81] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):pp. 649–673, 1987.

[82] G. Leurent and P. Q. Nguyen. How risky is the random-oracle model? In Halevi [61], pages 445–464.

[83] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[84] V. Lyubashevsky. On random high density subset sums. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 12, 2005.

[85] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2006.

[86] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably Secure FFT Hashing. *2nd NIST Cryptographic Hash Function Workshop*, 2006.

[87] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In K. Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008.

[88] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[89] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[90] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *Information Theory, IEEE Transactions on*, 24(5):525 – 530, sep 1978.

[91] R. C. Merkle. A certified digital signature. In Brassard [28], pages 218–238.

[92] D. Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, Mar. 2001. Preliminary version in FOCS 1998.

[93] 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.

[94] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.

[95] L. Minder and A. Sinclair. The extended k-tree algorithm. In C. Mathieu, editor, *SODA*, pages 586–595. SIAM, 2009.

[96] L. Minder and A. Sinclair. The extended k-tree algorithm. *Journal of Cryptology*, pages 1–34, 2011.

[97] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–519, 1985.

[98] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Math. Comput.*, 29:183–205, 1975.

[99] National Institute of Standards and Technology. FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2. Technical report, Aug. 2002.

[100] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA–3) family. *Federal Register*, 72(212):62212–62220, Nov 2007.

[101] P. Q. Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 1999.

[102] P. Q. Nguyen, editor. *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*. Springer, 2006.

[103] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control Inf. Theory*, 15:159–166, 1986.

[104] R. Overbeck and N. Sendrier. Code-based cryptography. Bernstein, Daniel J. (ed.) et al., Post-quantum cryptography. First international workshop PQCrypto 2006, Leuven, The Netherland, May 23–26, 2006. Selected papers. Berlin: Springer. 95-145 (2009)., 2009.

[105] 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.

[106] C. Petit, K. Lauter, and J.-J. Quisquater. Cayley Hashes: A Class of Efficient Graph-based Hash Functions. 2007.

[107] C. Petit, K. Lauter, and J.-J. Quisquater. Full Cryptanalysis of LPS and Morgenstern Hash Functions. 2008.

[108] C. Petit and J.-J. Quisquater. Preimages for the Tillich-Zémor hash function. In *SAC2010 Selected Areas in Cryptography*, 8 2010.

[109] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory*, 24:106–110, 1978.

[110] D. Pointcheval. The composite discrete logarithm and secure authentication. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2000.

[111] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Math. Comput.*, 32:918–924, 1978.

[112] J. M. Pollard. Factoring with cubic integers. Lenstra, A. K. (ed.) et al., The development of the number field sieve. Berlin: Springer-Verlag. Lect. Notes Math. 1554, 4-10 (1993)., 1993.

[113] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[114] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[115] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), Apr. 1992. Updated by RFC 6151.

[116] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[117] P. Rogaway. Formalizing human ignorance. In Nguyen [102], pages 211–228.

[118] 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.

[119] B. K. Roy and N. Sendrier, editors. *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*, volume 5922 of *Lecture Notes in Computer Science*. Springer, 2009.

[120] M.-J. O. Saarinen. Security of VSH in the real world. In Barua and Lange [13], pages 95–103.

[121] M.-J. O. Saarinen. Linearization attacks against syndrome based hashes. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2007.

[122] J. Šarinay. Interpreting hash function security proofs. In S.-H. Heng and K. Kurosawa, editors, *ProvSec*, volume 6402 of *Lecture Notes in Computer Science*, pages 119–132. Springer, 2010.

[123] J. Šarinay. Faster and smoother - VSH revisited. In U. Parampalli and P. Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2011.

[124] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[125] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[126] C.-P. Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *EUROCRYPT*, pages 1–12, 1995.

[127] C.-P. Schnorr and S. Vaudenay. The black-box model for cryptographic primitives. *J. Cryptology*, 11(2):125–140, 1998.

[128] R. Schroeppel and A. Shamir. A T=$O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

[129] A. Shallue. An improved multi-set algorithm for the dense subset sum problem. van der Poorten, Alfred J. (ed.) et al., Algorithmic number theory. 8th international symposium, ANTS-VIII Banff, Canada, May 17–22, 2008 Proceedings. Berlin: Springer. Lecture Notes in Computer Science 5011, 416-429 (2008)., 2008.

[130] 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.

[131] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[132] N. P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.

[133] M. Stam. Blockcipher-based hashing revisited. In O. Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2009.

[134] W. Stein et al. *Sage Mathematics Software (Version 4.7.1)*. The Sage Development Team, 2011. `http://www.sagemath.org`.

[135] R. Steinwandt, M. Grassl, W. Geiselmann, and T. Beth. Weaknesses in the $\mathrm{SL}_2(\mathbb{F}_{2^n})$ hashing scheme. In M. Bellare, editor, *CRYPTO*, volume

1880 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2000.

[136] J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[137] M. Stevens, A. Sotirov, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. In Halevi [61], pages 55–69.

[138] J.-P. Tillich and G. Zémor. Group-theoretic hash functions. In G. D. Cohen, S. Litsyn, A. Lobstein, and G. Zémor, editors, *Algebraic Coding*, volume 781 of *Lecture Notes in Computer Science*, pages 90–110. Springer, 1993.

[139] J.-P. Tillich and G. Zémor. Hashing with SL_2. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994.

[140] J.-P. Tillich and G. Zémor. Collisions for the LPS Expander Graph Hash Function. In Smart [132], pages 254–269.

[141] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in lattices. *Math. Dept. Report 81—. 04. Univ. of Amsterdam*, 1981.

[142] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[143] S. Vaudenay. Cryptanalysis of the Chor-Rivest cryptosystem. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 1998.

[144] S. Vaudenay. Cryptanalysis of the Chor-Rivest cryptosystem. *J. Cryptology*, 14(2):87–100, 2001.

[145] S. Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[146] D. Wagner. A Generalized Birthday Problem. Full version, `http://www.eecs.berkeley.edu/~daw/papers/genbday-long.ps`.

[147] D. Wagner. A Generalized Birthday Problem. In Yung [151], pages 288–303.

[148] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[149] X. Wang and H. Yu. How to break MD5 and other hash functions. In Cramer [43], pages 19–35.

[150] B.-Y. Yang, C.-H. O. Chen, D. J. Bernstein, and J.-M. Chen. Analysis of QUAD. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 290–308. Springer, 2007.

[151] 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.

[152] G. Zémor. Hash functions and graphs with large girths. In *EUROCRYPT*, pages 508–511, 1991.

[153] G. Zémor. Hash functions and Cayley graphs. *Des. Codes Cryptography*, 4(4):381–394, 1994.

Curriculum Vitae

Juraj Šarinay
Born 8 April 1983

Education

Sep 2007 – Dec 2011	PhD in Computer Science School of Computer and Communication Sciences École Polytechnique Fédérale de Lausanne
Sep 2001 – May 2006	Master Degree in Mathematics Faculty of Mathematics, Physics and Informatics Comenius University in Bratislava, Slovakia

Work Experience

Sep 2007 – Dec 2011	Assistant Laboratory For Cryptologic Algorithms École Polytechnique Fédérale de Lausanne
Oct 2003 – Aug 2007	Malware Researcher Virus Laboratory ESET spol. s r.o., Bratislava, Slovakia