5-2015

Some applications of noncommutative groups and semigroups to information security

Lisa Bromberg
Graduate Center, City University of New York

Some applications of noncommutative groups and semigroups to information security

by

Lisa Bromberg

A dissertation submitted to the Graduate Faculty in Mathematics in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2015

This manuscript has been read and accepted by the Graduate Faculty in Mathematics in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

	Professor Vladimir Shpilrain
Date	Chair of Examining Committee
	Professor Linda Keen
Date	Executive Officer

Professor Vladimir Shpilrain

Professor Victor Pan

Professor Melvyn Nathanson

Supervisory Committee

Abstract

Some applications of noncommutative groups

and semigroups to information security

by

Lisa Bromberg

Adviser: Professor Vladimir Shpilrain

We present evidence why the Burnside groups of exponent 3 could be a good candidate for a platform group for the HKKS semidirect product key exchange protocol. We also explore hashing with matrices over $\mathrm{SL}_2(\mathbb{F}_p)$, and compute bounds on the girth of the Cayley graph of the subgroup of $\mathrm{SL}_2(\mathbb{F}_p)$ for specific generators A, B. We demonstrate that even without optimization, these hashes have comparable performance to hashes in the SHA family.

Acknowledgments

I would like to express my deepest gratitude to my thesis adviser, Vladimir Shpilrain, for his continued support and encouragement throughout this process. He was a patient mentor from whom I have learned a lot of mathematics. I am also very grateful to Delaram Kahrobaei, for introducing me to noncommutative cryptography and supporting me in many ways as a graduate student. Lastly, I'd like to thank Victor Pan and Melvyn Nathanson for serving on my defense committee.

Contents

Chapter 1

Introduction to Cryptography

1.1 Public Key Cryptography

The aim of cryptography is to protect information from being stolen or modified by an adversary. In modern cryptography, specific security goals are achieved with the design of algorithms and also using the known computational hardness of certain mathematical problems.

There are currently two main classes of cryptographic primitives: *public-key (assymetric)* and *symmetric-key*. Symmetric-key algorithms are older, and in fact can be traced back to at least the time of Julius Caesar. In symmetric-key ciphers, knowledge of the encryption key is usually equivalent (or equal) to knowledge of the decryption key. Because of this, participating parties need to agree on a shared secret key before communicating through an open channel.

Public-key cryptography is a relatively young area of mathematics, but it has been a very active area of research since its inception in 1976, with a seminal paper of Diffie and Hellman [9]. In public-key algorithms, there are two separate keys: a public key that is published and a private key which each user keeps secret. Knowledge of the public key does

not imply knowledge of the private key with any efficient computation. In fact, the public key is generated from the private key using a *one-way* function, with a *trapdoor*, which is a function that is easy (i.e. polynomial-time with respect to the complexity of an input) to compute, but hard (no visible (probabilistic) polynomial-time algorithm on "most" inputs) to invert the image of a random input without special information; the special information is the above-mentioned "trapdoor". A well-known example of public-key encryption is the RSA cryptosystem, whose one-way function is the product of two large primes p, q. If p and q are known, then it is easy to compute their product, but it is hard to factor a large number into its prime factors.

Since public-key cryptosystems are more computationally costly than symmetric algorithms, some modern cryptosystems rely on an assymetric cipher to produce a session key, and then proceed with symmetric encryption for the remainder of the session.

1.2 Problems common in cryptography

Most classic cryptographic primitives involve the use of finite abelian (commutative) groups. The two main problems security in this setting relies on are factoring and the discrete logarithm. Our current technology keeps these problems hard, but there are efficient quantum algorithms which solve both of these. This motivated research in expanding cryptographic primitives which are based on other areas of mathematics. In particular, there is a lot of research in cryptography based on noncommutative groups.

Some problems used in noncommutative cryptography are based on combinatorial group theory. For more details on those problems, see Myasnikov, Shpilrain and Ushakov [38]. The problems from group theory we are concerned with fall into three main types: decision, witness and search problems.

Decision problems Given an object $\mathcal{O}$ and a property $\mathcal{P}$, determine whether $\mathcal{O}$ has

property $\mathcal{P}$.

Witness problems Given an object $\mathcal{O}$ with a property $\mathcal{P}$, find a proof of the fact that $\mathcal{O}$ has property $\mathcal{P}$.

Search problems Given an object $\mathcal{O}$ with a property $\mathcal{P}$, find something 'material' establishing the property $\mathcal{P}$. (An example we are particularly interested in is the *conjugacy search problem*; see below.)

Note that search problems are in fact a special case of witness problems.

We now introduce several commonly used problems in noncommutative cryptography. Let G be a finitely presented group with presentation $\langle X \mid R \rangle$, where X is the set of generators and R is the set of relators. We let 1 denote the identity, and $y^x := x^{-1}yx$ for all $x, y \in G$.

Word Problem Given a group G and an element $g \in G$, determine whether $g =_G 1$. The *word search problem (WSP)* is: given an element $g \in G$ such that $g =_G 1$, find a presentation of g as a product of conjugates of defining relators and their inverses.

Membership Problem Given a subgroup $H \leq G$ and an element $g \in G$, determine whether $g \in H$. The *membership search problem (MSP)* is: given a subgroup $H \leq G$ generated by $h_1, \ldots, h_k$ and an element $h \in H$, find a presentation of h in terms of the generators $h_1, \ldots, h_k$. A cryptosystem of Shpilrain and Zapata [50] makes use of the subgroup membership search problem.

Conjugacy Problem Given $g, h \in G$, determine whether there exists $x \in G$ such that $g^x = h$. The *conjugacy search problem (CSP)* is: given $g, h \in G$ such that they are conjugate, find an $x \in G$ such that $g^x = h$.

There are variants of the conjugacy search problem, such as the simultaneous conjugacy search problem, which are employed in the Anshel–Anshel–Goldfeld key-exchange protocol [1]. The conjugacy search problem has been used for several other cryptographic protocols, such as the noncommutative Diffie–Hellman key exchange [29], the noncommutative

El-Gamal encryption [24], the noncommutative Cramer–Shoup key exchange [23] and noncommutative digital signatures [25].

1.3 Platform groups

For any noncommutative cryptographic primitive, there are several requirements we have of the platform group used. For more details on these requirements, see Myasnikov, Shpilrain and Ushakov [38, 39].

First, the group G should be recursively presented; this allows us to encode the group in a computer system.

We also require that the word problem in G can be solved efficiently so that the parties involved can know they have the same common key. Normal form can be used to disguise elements of G, and also implies that the word problem can be solved.

While we do want the word problem to be solvable, we require that some algorithm problem for G have no efficient solution. In general, this is hard to prove, so in practice we expect only that one or more search problems for the group G has been well-studied and that no efficient solution has been found.

To ensure a large enough key space, we require that G have exponential or intermediate (as opposed to polynomial) growth rate. The *growth function* of a group G tells us the number of elements of G with length n. Having nonpolynomial growth makes the key space large, which prevents a brute force attack.

Less formal and articulated considerations for the platform group include things like the ease of implementation of G (we should not pick a platform group which is hard for participants to compute with).

Chapter 2

Ramifications of the Diffie–Hellman key exchange protocol

In this chapter we talk about a key exchange protocol introduced by Habeeb, Kahrobaei, Koupparis and Shpilrain [16], which we will refer to as the HKKS protocol. It is an extension to nonabelian groups of the Diffie–Hellman (DH) key exchange, using the semidirect product and extensions by automorphisms.

2.1 Diffie–Hellman key exchange protocol

In a seminal paper, Diffie and Hellman introduced the first public key exchange protocol [9]. Originally, the protocol uses the multiplicative group of integers modulo a prime p, $G = (\mathbb{Z}/p\mathbb{Z})^*$, and a primitive element g of G (i.e. g generates G). We can explain the protocol using an arbitrary cyclic group.

Algorithm 1 (DH key exchange protocol)**.** (1) Alice and Bob agree on a (public) finite cyclic group G and a generating element g of G.

(2) Alice picks a random number a and sends g^a to Bob.

(3) Bob picks a random number b and sends g^b to Alice.

(4) Alice computes $K_A := (g^b)^a = g^{ba}$.

(5) Bob computes $K_B := (g^a)^b = g^{ab}$.

Since $ab = ba$, Alice and Bob have a shared secret key.

The security of the Diffie–Hellman protocol is based on the difficulty of the *Diffie–Hellman problem*.

Definition 1 (Diffie–Hellman problem)**.** Given a finite cyclic group G, a generating element g, and g^a, g^b, where $a, b \in \mathbb{N}$, find g^{ab}.

A stronger problem is the *discrete logarithm problem*.

Definition 2 (Discrete logarithm problem)**.** Given a finite cyclic group G, a generating element g, and g^a where $a \in \mathbb{N}$, find a.

It is evident that an efficient algorithm which solves the discrete logarithm problem will also solve the Diffie–Hellman problem: use the algorithm to recover a, b from g^a, g^b respectively. Then simply compute g^{ab}. However, it is not currently known whether the two problems are equivalent.

2.2 Ko-Lee key exchange

One of the possible generalizations of the discrete logarithm problem is the *conjugacy search problem*: given two elements a, b of a group G, and the knowledge that there exists $x \in G$ such that $a^x := x^{-1}ax = b$, find at least one particular element x which conjugates a to b. The (alleged) computational difficulty of solving this problem in groups such as braid groups has been used in several cryptosystems. We outline one of those here, namely the Ko–Lee key exchange protocol.

Algorithm 2 (Ko–Lee key exchange protocol)**.** Let G be a nonabelian group, and recall that the notation a^x denotes conjugation: $a^x = x^{-1}ax$. An element $w \in G$ is public. Two subgroups A, B of G such that $[A, B] = 1_G$ are also public (i.e. $ab = ba$ for all $a \in A, b \in B$).

(1) Alice chooses an element $a \in A$ and sends w^a to Bob, while keeping a private.

(2) Bob chooses an element $b \in B$ and sends w^b to Alice, while keeping b private.

(3) Alice computes $K_A = (w^b)^a = w^{ba}$.

(4) Bob computes $K_B = (w^a)^b = w^{ab}$.

Since $ab = ba$ for all $a \in A, b \in B$, we have that $K_A = K_B$ is the shared secret key K.

Their proposed system was based on using *braid groups*. The n–braid group Br_n is an infinite noncommutative group of n–braids defined for each positive integer n. A geometric interpretation is as follows: an n–braid is a set of n disjoint strands, all of which are attached to two horizontal bars at the top and the bottom such that each strand always heads downward as one walks along the strand from top to bottom. See [29, Section 2] for a more detailed definition.

As these groups play a role in areas of math including low-dimensional topology, combinatorial group theory and representation theiry, they are well-studied groups. For an adversary to find the shared secret key K, they could solve the conjugacy search problem, which Ko–Lee describe as being mathematically hard.

However, in [49], Shpilrain and Ushakov explain that it is sufficient for an adversary to find a_1, a_2 such that $a_1 w a_2 = a^{-1}wa$ and b_1, b_2 such that $b_1 w b_2 = b^{-1}wb$, since if we also suppose that a_1, a_2 commute with all $b \in B$, then we have

$$a_1 b_1 w a_2 b_2 = a_1 b^{-1} w b a_2 = b^{-1} a w a b = b^{-1} a^{-1} w a b = K.$$

Note that a_1, a_2, b_1, b_2 have nothing to do with Alice and Bob's private elements. In particular, it is not necessary for the adversary to solve the conjugacy search problem, but rather it is sufficient to solve the apparently easier *decomposition problem*, which has the conjugacy search problem as a special case.

2.3 The HKKS key exchange protocol

Since it is now a prevalent opinion that the conjugacy search problem is unlikely to provide a sufficient level of security if a noncommutative group is used as the platform, we investigate a different direction to take. In this section we will describe the key exchange protocol introduced in [16]. But first we will give some background on the semidirect product.

2.3.1 Semidirect Product

Definition 3. Let G, H be two groups, let $\mathrm{Aut}(G)$ be the group of automorphisms of G and let $\varphi\colon\ H \to \mathrm{Aut}(G)$ be a homomorphism. Then the *semidirect product* of G and H is the set

$$G \rtimes_\varphi H = \{(g, h) \mid g \in G, h \in H\}$$

with the group operation

$$(g, h) \cdot (g', h') = (g^{\varphi(h')} \cdot g', h \cdot h').$$

Note that $g^{\varphi(h')}$ denotes $\varphi(h')(g)$ and $h \cdot h'$ denotes composition with h applied first.

If $H = \mathrm{Aut}(G)$, then the corresponding semidirect product is called the *holomorph* of G, and is the set of all (g, ϕ) with $g \in G, \phi \in \mathrm{Aut}(G)$ with group operation $(g, \phi) \cdot (g', \phi') = (\phi'(g) \cdot \phi, \phi \cdot \phi')$.

A special case of the HKKS construction is *extension by automorphisms* where we do not use the whole group $\mathrm{Aut}(G)$ but only a cyclic subgroup of it, which is generated by a fixed $\phi \in \mathrm{Aut}(G)$. The resulting object Γ is also a group. Since every automorphism that we are concerned with now is an element of $\langle\phi\rangle$, the group operation becomes

$$(g, \phi^r)(h, \phi^s) = (\phi^s(g) \cdot h, \phi^{r+s}).$$

In particular, to calculate exponents of an element of Γ, we have

$$\begin{aligned}(g, \phi)^m &= (g, \phi) \cdot (g, \phi) \cdot (g, \phi)^{m-2} \\ &= (\phi(g) \cdot g, \phi^2) \cdot (g, \phi) \cdot (g, \phi)^{m-3} \\ &= (\phi^2(g) \cdot \phi(g) \cdot g, \phi^3) \cdot (g, \phi) \cdot (g, \phi)^{m-4} \\ &= \dots \\ &= (\phi^{m-1}(g) \cdot \phi^{m-2}(g) \cdots \phi(g) \cdot g, \phi^m)\end{aligned}$$

This construction works just as well with a semigroup G and and endomorphism ϕ instead of a group and an automorphism. In that case, the resulting object Γ is a semigroup instead.

2.3.2 The protocol

The protocol works as follows [16]. Let G be a public (semi)group, and let $g \in G$ and $\phi \in \mathrm{Aut}(G)$ be public as well. Alice chooses a private integer m and Bob chooses a private integer n.

(1) Alice computes $(g, \phi)^m = (\phi^{m-1}(g) \cdots \phi(g) \cdot g, \phi^m)$ and sends only the first component to Bob; call that component a.

(2) Bob computes $(g, \phi)^n = (\phi^{n-1}(g) \cdots \phi(g) \cdot g, \phi^n)$ and sends only the first component to

Alice; call it b.

(3) Alice computes $(b, x) \cdot (a, \phi^m) = (\phi^m(b) \cdot a, x \cdot \phi^m)$. Her key is $K_A = \phi^m(b) \cdot a$.

(4) Bob computes $(a, y) \cdot (b, \phi^n) = (\phi^n(a) \cdot b, y \cdot \phi^n)$. His key is $K_B = \phi^n(a) \cdot b$.

Since $K_A = (g, \phi)^{m+n} = K_B$, Alice and Bob have a shared secret key.

Note that use of the 'square-and-multiply' method greatly reduces the computational cost of implementing the protocol. The exact cost, however, depends on the platform group being used.

2.4 Platforms used for the HKKS protocol

2.4.1 Multiplicative group of integers modulo p

A simple application of the HKKS protocol is to use the multiplicative group of integers modulo a prime p. The public endomorphism ϕ is selected by choosing $k > 1$ so that for each $h \in (\mathbb{Z}/p\mathbb{Z})^\times$, we have $\phi)h) = h^k$. Note that if k is relatively prime to $p-1$, then ϕ is in fact an automorphism. Alice and Bob choose private m, n as in the Diffie–Hellman protocol.

Note that if $g \in (\mathbb{Z}/p\mathbb{Z})^\times$, then

$$\begin{aligned}(g, \phi)^m &= (\phi^{m-1}(g) \cdot \phi^{m-2}(g) \cdots \phi(g) \cdot g, \phi^m) \\ &= (g^{k^{m-1}} \cdot g^{k^{m-2}} \cdots g^k \cdot g, \phi^m) \\ &= (g^{k^{m-1}+\cdots+k+1}, \phi^m) \\ &= (g^{\frac{k^m-1}{k-1}}, \phi^m).\end{aligned}$$

So, Alice sends $\frac{k^m-1}{k-1}$ to Bob, and Bob sends $\frac{k^n-1}{k-1}$ to Alice. The shared secret key is the first component of $(g, \phi)^{m+n}$ which is $g^{\frac{k^{m+n}-1}{k-1}}$.

A direct attack of this protocol would have the adversary (generally referred to as Eve) try to recover m, n. To acquire either, Eve has to recover $\frac{k^m-1}{k-1}$ from $g^{\frac{k^m-1}{k-1}}$, and then recover m from $\frac{k^m-1}{k-1}$. This amounts to solving the discrete logarithm problem twice.

Instead, Eve can view this as an analog of the Diffie–Hellman problem: recover the shared secret $g^{\frac{k^{m+n}-1}{k-1}}$ from the triple $\left(g, g^{\frac{k^m-1}{k-1}}, g^{\frac{k^n-1}{k-1}}\right)$. Since g, k are public, it is enough to recover $g^{k^{m+n}}$ from the triple (g, g^{k^m}, g^{k^n}). This is precisely the standard Diffie–Hellman problem (see Definition 1).

This instantiation of the protocol is the equivalent of the classic Diffie–Hellman protocol, so breaking the HKKS protocol for any cyclic group would imply breaking the Diffie–Hellman protocol.

2.4.2 Matrices over a group ring

Another platform investigated in [16] uses a semigroup and an inner automorphism.

Definition 4. Let G be a group, written multiplicatively, and let R be a ring. The *(semi)group ring* of G over R, written $R[G]$, is the set of formal linear combinations of elements of G with coefficients in R:

$$\sum_{g \in G} r_i g,$$

where $r_i \in R$, and all but finitely many r_i are zero.

The particular platform suggested by the authors in [16] is the semigroup G of 3×3 matrices over the group ring $\mathbb{Z}_7[A_5]$, where A_5 is the alternating group on 5 elements. We denote the semigroup by $G = \mathrm{GL}_3(\mathbb{Z}_7[A_5])$.

The semigroup G is extended by an inner automorphism ϕ_H, which is conjugation by a matrix $H \in G$, i.e. $\phi_H(M) = H^{-1}MH$. Both M and H are public. Note that for all $M \in G$ and for all integers $k \geq 1$, $\phi_H^k(M) = H^{-k}MH^k$.

In the semidirect product, exponentiating an element gives

$$\begin{aligned}(M, \phi_H)^m &= (\phi_H^{m-1}(M) \cdot \phi_H^{m-2}(M) \cdots \phi_H(M) \cdot M, \phi_H^m) \\ &= (H^{-m+1} M H^{m-1} \cdots H^{-2} M H^2 \cdot H^{-1} M H, \phi_H^m) \\ &= H^{-m}(HM)^m.\end{aligned}$$

So, Alice sends the matrix $H^{-m}(HM)^m$ to Bob, and Bob sends the matrix $H^{-n}(HM)^n$ to Alice; the shared secret key is $H^{-(m+n)}(HM)^{m+n}$.

Eve can try to recover private m from $H^{-m}(HM)^m$. This problem appears to be hard. In the case that $H = \mathrm{Id}$, we have the analog of the discrete logarithm problem over $\mathbb{Z}_7[A_5]$, i.e. to recover m given M and M^m. Using statistical experiments, it was shown [26] that for a random matrix M, that M^m are indistinguishable from random matrices.

Cryptanalysis

The security of this protocol is based on the assumption that, given $M \in M_3(\mathbb{F}_7[A_5])$, $H \in \mathrm{GL}_3(\mathbb{F}_7[A_5])$, $A = H^{-m}(HM)^m$ and $B = H^{-n}(HM)^n$, it is hard to compute the matrix $H^{-n-m}(HM)^{n+m}$.

In [30], Kreuzer, Myasnikov and Ushakov show that the above problem can be solved using the fact that H is invertible. In fact, any solution to the system

$$\begin{cases} LA = R \\ LH = HL \\ RHM = HMR \\ L \text{ is invertible} \end{cases}$$

with unknown matrices L, R immediately gives the shared key as $L^{-1}BR$. To solve the

system, find a solution satisfying the first three equations, and check if L is invertible. With high probability this process will lead to a solution in a small number of tries.

Another independent attack was described by Romankov in [46]. In particular, he shows that the shared secret key can be computed in the case when the underlying (semi)group G is a multiplicative subgroup of a finite-dimensional algebra $\mathcal{A}$ over a field $\mathbb{F}$ and the endomorphism ϕ is extended to an endomorphism of the underlying vector space V of $\mathcal{A}$. We describe the method below.

Using Gaussian elimination, find a maximal linearly independent subset L of the set $\{a_0, a_1, \ldots, a_k, \ldots\}$, where $a_0 = g, a_k = \phi^{k-1}(g) \cdots \phi(g) \cdot g$ for $k \geq 1$. In fact, suppose $\{a_0, \ldots, a_k\}$ is linearly independent but a_{k+1} can be presented as an $\mathbb{F}$–linear combination of the a_i, $0 \leq i \leq k$, i.e. $a_{k+1} = \sum_{i=0}^{k} \lambda_i a_i$ with $\lambda_i \in \mathbb{F}$. Suppose by induction that a_{k+j} can be written as an $\mathbb{F}$–linear combination of the a_i, $0 \leq i \leq k$ for every $j \leq t-1$, i.e.

$$a_{k+t-1} = \sum_{i=0}^{k} \mu_i a_i, \quad \mu_i \in \mathbb{F}.$$

Then

$$\begin{aligned} a_{k+t} &= \phi(a_{k+t-1}) \cdot g \\ &= \sum_{i=0}^{k} \mu_i \phi(a_i) \cdot g \\ &= \sum_{i=0}^{k} \mu_i a_{i+1} \\ &= \mu_k \lambda_0 a_0 + \sum_{i=0}^{k-1} (\mu_i + \mu_k \lambda_{i+1} a_{i+1}). \end{aligned}$$

Thus $L = \{a_0, \ldots, a_k\}$. In particular, we can effectively compute

$$a_n = \sum_{i=0}^{k} \eta_i a_i, \quad \eta_i \in \mathbb{F}.$$

Then

$$a_{m+n} = \phi^m(a_n) \cdot a_m = \sum_{i=0}^{k} \eta_i \phi^m(a_i) \cdot a_m = \sum_{i=0}^{k} \eta_i \phi^i(a_m) \cdot a_i;$$

in particular we have the shared secret key $K = a_{m+n}$. Therefore, there is a polynomial-time algorithm to find the shared secret key K from the public data.

In light of this, in [27], Kahrobaei, Lam and Shpilrain propose the following countermeasure to prevent this attack. Since the attack splits the public key A into a product of two matrices which act as H^{-m} and $(HM)^m$, then if M is not invertible, the *annihilator* of HM,

$$\mathrm{Ann}(HM) := \{K \in M_3(\mathbb{F}_7[A_5]) \mid K \cdot HM = O\}$$

(where O here denotes the zero matrix), is nontrivial. Since $m, n > 0$, then adding $O_A, O_B \in \mathrm{Ann}(HM)$ to the public keys A and B changes those keys, but not the shared key.

The idea behind this modification is that now A cannot be split into a product of two matrices to move one to the left-hand side. However, in [10], Ding, Miasnikov and Ushakov show this is incorrect and the same attack can be applied. Using the fact that the annihilator

is a left ideal and H is invertible, then any solution of the system

$$\begin{cases} LA = R + Z \\ LH = HL \\ R \cdot HM = HM \cdot R \\ Z \cdot HM = O \\ L \text{ is invertible} \end{cases}$$

with unknown matrices L, R, Z gives the shared secret key as the product $L^{-1}BR$ (it is essential that H is invertible).

2.4.3 Matrices over a Galois field

In her PhD thesis [32], Ha Lam proposes a countermeasure to the attack described above using a more complex endomorphism rather than an inner automorphism. This requires a change of platform group; she uses matrices over a Galois, or finite, field. In particular, Lam uses binary fields, which are finite fields of order 2^m, denoted $\mathbb{GF}(2^m)$. Binary fields are widely used in cryptography, and well-studied algorithms make computation very fast. For background on binary fields, we refer the reader to [32].

Lam suggests using the semigroup of 2×2 matrices over the Galois field $\mathbb{GF}(2^t)$ as a platform for the HKKS key exchange. In fact, she uses an extension of G by an endomorphism ϕ which is a composition of a conjugation by a matrix $H \in \mathrm{GL}_2(\mathbb{GF}(2^t))$ with the endomorphism ψ which raises each entry of a given matrix to the power 4. We apply ψ first, followed by conjugation. So, with public elements M and H, we have $\phi(M) = H^{-1}\psi(M)H$.

So, for any $M \in G$ and for all integers $k \geq 1$,

$$\begin{aligned}\phi^k(M) &= \phi^{k-1}(H^{-1}\psi(M)H) \\ &= \phi^{k-2}(\phi(H^{-1}\psi(M)H)) = \phi^{k-2}(H^{-1}\psi(H^{-1}\phi^2(M)\cdots\psi(H)H) \\ &\vdots \\ &= H^{-1}\psi(H^{-1}\cdots\psi^{k-1}(H^{-1})\psi^k(M)\psi^{k-1}(H)\cdots\psi(H)H.\end{aligned}$$

Then the HKKS protocol works as follows.

(1) Alice and Bob agree on a degree k for the Galois field and public matrices $M \in G$ and $H \in \mathrm{GL}_2(\mathbb{GF}(2^t))$. Alice and Bob select their respective private positive integers m and n.

(2) Alice computes $(M, \phi)^m$ and sends the first component, A, to Bob.

(3) Bob computes $(M, \phi)^n$ and sends the first component, B, to Alice.

(4) Alice's secret key is $K_A = \phi^m(B) \cdot A$, which is the first component of $(M, \phi)^{m+n}$. Similarly, Bob's secret key is $K_B = \phi^n(A) \cdot B$, which is again the first component of $(M, \phi)^{m+n}$.

Since $K_A = K_B$, this is the shared secret key K.

The security assumption of the protocol is that it is hard to recover K from the matrices H, M, A, B with A and B as defined above. First, considering only $\phi = H^{-1}MH$, note that in the simplest case ($\phi = \mathrm{Id}$) yields the analog of the discrete logarithm problem for matrices over $\mathbb{GF}(2^t)$, i.e. to recover m given M and M^m. Pollard's rho method and Shank's baby-step-giant-step method are standard techniques for attacking the discrete logarithm problem (see Sections 3.2 and 3.3 respectively), and are based on the difficulty of detecting cycles. Here, looking for a cycle means finding i, j such that $x^{-i}y^i = x^{-j}y^j$, where $x = H$,

$y = HM$. If x and y are both invertible, then this is really the problem of finding k such that $x^k = y^k$, so instead of detecting a cycle, it is a problem of finding an intersection of two independent cycles. This is apparently a more difficult problem.

Another possible attack involves the determinants of the public matrices H, M, A, B, since recovering m can be reduced to the discrete log problem for $(\det(M), (\det(M))^n)$. To prevent this attack, simply choose M to have determinant either 0 or 1.

The implementation with this platform group was to use $\mathbb{GF}(2^{127})$ and $\mathbb{GF}(2^{571})$. For details on the efficiency, see [32].

Cryptanalysis

A similar attack as the one on this protocol using matrices over a group ring can be applied here, as described by Ding, Miasnikov and Ushakov in [10]. Noting that the map $x \overset{\tau}{\mapsto} x^4$ defined on $\mathbb{GF}(2^{127})$ can be seen as a square of the Frobenius automorphism, and induces the automorphism ψ of $M_2(\mathbb{GF}(2^{127}))$ of order 127.

Recall that $\phi(M) = H^{-1}\psi(M)H$ for all matrices $M \in M_2(\mathbb{GF}(2^{127}))$, and for all $k \in \mathbb{N}$, we have

$$\phi^k(M) = \prod_{i=0}^{k-1} \psi^i(H^{-1}) \cdot \psi^k(M) \cdot \prod_{i=k-1}^{0} \psi^i(M).$$

With ϕ given in this form, Alice's public key $A = \phi^{m-1}(M) \cdots \phi(M)M$ is of the form

$$\begin{aligned}
&\left(\prod_{i=0}^{m-1} \psi^i(H^{-1}) \cdot \psi^m(M) \cdot \prod_{i=m-1}^{0} \psi^i(H)\right) \cdot \left(\prod_{i=0}^{m-2} \psi^i(H^{-1}) \cdot \psi^{m-1}(M) \cdot \prod_{i=m-2}^{0} \psi^i(H)\right) \\
&\qquad \cdots H^{-1}\psi(M)H \cdot M \\
&= \left(\prod_{i=0}^{m-1} \psi^i(H^{-1}) \cdot \psi^m(M)\right) \psi^{m-1}(H)\psi^{m-1}(M) \cdot psi^{m-2}(H)\psi^{m-2}(M) \\
&\qquad \cdots \psi(H)\psi(M) \cdot HM \\
&= \left(\prod_{i=0}^{m} \psi^i(H^{-1})\right) \cdot \left(\prod_{i=m}^{0} \psi^i(HM)\right).
\end{aligned}$$

Since $|\psi| = 127$, we can divide $m = 127q + r$ and write Alice's key as

$$A = \left(\prod_{i=0}^{126} \psi^i(H^{-1})\right)^q \cdot \left(\prod_{i=0}^{r} \psi^i(H^{-1})\right) \cdot \left(\prod_{i=r}^{0} \psi^i(HM)\right) \cdot \left(\prod_{i=126}^{0} \psi^i(HM)\right)^q.$$

Bob's public key can be written similarly, using the fact that $n = 127s + t$:

$$A = \left(\prod_{i=0}^{126} \psi^i(H^{-1})\right)^s \cdot \left(\prod_{i=0}^{t} \psi^i(H^{-1})\right) \cdot \left(\prod_{i=t}^{0} \psi^i(HM)\right) \cdot \left(\prod_{i=126}^{0} \psi^i(HM)\right)^s.$$

Now, for each $0 \leq r \leq 126$, we can try to solve the system of equations

$$\begin{cases} L \cdot A = \left(\prod_{i=0}^{r} \psi^i(H^{-1})\right) \cdot \left(\prod_{i=r}^{0} \psi^i(HM)\right) \cdot R, \\ L \cdot \prod_{i=0}^{126} \psi^i(H^{-1}) = \prod_{i=0}^{126} \psi^i(H^{-1}) \cdot L, \\ R \cdot \prod_{i=126}^{0} \psi^i(HM) = \prod_{i=126}^{0} \psi^i(HM) \cdot R, \\ L \text{ is invertible.} \end{cases}$$

If the pair (L, R) satisfies the above system, then $L^{-1}BR$ is the shared key, i.e. the same linear algebra attack does in fact work on the HKKS protocol using matrices over a Galois field as the platform group.

Under the assumption of the linearity of the platform group, even the proposed improvements by Lam are shown to be susceptible to the linear decomposition attack of Romankov described in [46]. In contrast to the linear algebra attack described above, the linear decomposition attack is fairly simple.

Consider the linear space W which is generated by all elements of the form $H^k(HM)^l$, where $k, l \in \mathbb{N}$, with effectively computed basis $e_1, \ldots, e_t$ (note that $t \leq 4$.) Since every matrix is a root of a characteristic polynomial of degree 2, one can choose basis elements of the form $e_i = H^{k_i}(HM)^{l_i}$, $l_i \in \{0, 1\}$, $i = 1, \ldots, t$. Then a_n, a_m can be effectively computed,

and from this, one can effectively compute the shared secret key $K = a_{m+n}$.

Even when Lam describes an altered protocol to avoid the linear algebra attack, this altered protocol is still vulnerable to the linear decomposition attack. Consider the linear space W generated by elements of the form $H^{-k}(HM)^k$, and note that $a_m, a_n \in W$. Let U denote the annihilator space of HM consisting of all matrices $A \in M_2(\mathbb{GF}(2^{127}))$ such that $A \cdot (HM) = 0$. Note that Alice's and Bob's private matrices lie in U. Let $Z = W + U$ with basis $\{e_1, \ldots e_l, f_1, \ldots, f_t\}$ with $e_i \in W$, $f_j \in U$ and $e_i = H^{-k_i}(HM)^{k_i}$. Then one can effectively compute a_n and hence the shared secret key $K = a_{m+n}$.

Note that in all instances of this cryptanalysis, the basis is constructed one time offline, and we do not need to look for an invertible solution to a system of linear equations. Note also that we do not need to compute m and/or n to recover the shared secret key K.

On the other hand, this attack only works if an inner automorphism (or a minor variation thereof) is used for extension of the platform (semi)group because otherwise, the long product would not simplify to $H^{-k}(HM)^k$.

2.5 Burnside groups: a new platform for the HKKS key exchange

In light of the recent attacks on Habeeb, Kahrobaei, Koupparis and Shpilrain's and Lam's parameters, in this section we propose using *Burnside groups* as a platform for the HKKS key-exchange protocol. We begin with background on Burnside groups, and outline why they are a good choice for a platform group. We note that Burnside groups have been used in [3] as platforms for a very different cryptographic protocol.

2.5.1 Burnside groups

For a positive integer k, consider the class of groups for which all elements x satisfy $x^k = 1$. Such a group is said to be of *exponent* k. We consider the family of such groups called the *(free) Burnside groups of exponent* k, which are in some sense the 'largest.' The Burnside groups are determined by two parameters: the exponent k and the number of generators n, and are denoted $B(n, k)$.

Definition 5 ((Free) Burnside group)**.** For any $n, k \geq 0$, the *Burnside group of exponent* k *with* n *generators* is defined as

$$B(n,k) = \langle \{x_1, \ldots, x_n\}; \{w^k \mid \text{for all words } w \text{ over } x_1, \ldots, x_n\} \rangle.$$

For our cryptographic purposes, it is important that $B(n, k)$ be finite. The question of whether a given Burnside group is finite is known as the *Burnside problem*. For sufficiently large k, it is known that $B(n, k)$ is infinite [22]. For small exponents, it is known that $B(n, k)$ is finite for $k \in \{2, 3, 4, 6\}$. With the exception of $k = 2$, these are nontrivial results. For other small values of k, the question of finiteness remains open.

We are interested in Burnside groups of exponent 3. The main reasons are as follows: $k = 2$ would give the more familiar (and already studied) case $B(n, 2) = \bigoplus_n \mathbb{Z}_2$; it is convenient for k to be prime (hence eliminating $k = 4$ and $k = 6$); and perhaps most importantly, the structure of $B(n, 3)$ is much better understood in comparison to that of $k = 4, 6$. Hence, in what follows we will deal only with $B(n, 3)$ and denote it simply by B_n.

We will first review some important facts about Burnside groups. For more details we refer the reader to [3].

B_n is free In the category of groups of exponent 3, B_n is a free object on the set of generators of size n. That is, if G is any group such that $g^3 = 1$ for all $g \in G$, then for any set map $f\colon A \to G$ (where A is a generating set) there exists a unique homomorphism $\bar{f}\colon B_n \to G$

such that $\bar{f}(x_i) = f(x_i)$ for every $i \in [n]$. In other words, to define a homomorphism from B_n to G we need only define the function on A. Any such assignment will extend uniquely to a group homomorphism.

From this we can compute that there are $3^{n\left(r+\binom{r}{2}+\binom{r}{3}\right)}$ homomorphisms from B_n to B_r.

Normal form It is known that B_n is nilpotent of class 3, meaning that the smallest possible length of a central series for the group is 3. Even though B_n is nonabelian, each element has a unique normal form, as result of the order law $w^3 = 1$ for all $w \in B_n$. Precisely, each element of B_n can be written as an ordered sequence of (a subset of) generators (or their inverses[1]), appearing in lexicographical order, followed by (a subset of) the commutators of weight 2 (or their inverses), and finally by (a subset of) the commutators of weight 3 (or their inverses):

$$\prod_{i=1}^{n} x_i^{\alpha_i} \prod_{i<j} [x_i, x_j]^{\beta_{i,j}} \prod_{i<j<k} [x_i, x_j, x_k]^{\gamma_{i,j,k}}$$

where $\alpha_i, \beta_{i,j}, \gamma_{i,j,k} \in \{0, 1, -1\}$ for all $1 \leq i < j < k \leq n$. Note that $[x_i, x_j, x_k] = [[x_i, x_j], x_k]$.

From the above normal form, B_n has order $3^{n+\binom{n}{2}+\binom{n}{3}}$.

Abelianization of B_n It is also clear from the normal form that the abelianization $B_n/[B_n, B_n]$ of B_n is isomorphic to $\bigoplus_n \mathbb{Z}_3$, and the abelianization map ρ is efficiently computable, as it amounts to taking a prefix of the exponent list from the normal form:

$$\rho\colon \prod_{i=1}^{n} x_i^{\alpha_i} \prod_{i<j} [x_i, x_j]^{\beta_{i,j}} \prod_{i<j<k} [x_i, x_j, x_k]^{\gamma_{i,j,k}} \in B_n \mapsto (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_3^n.$$

Center of B_n The center $Z(B_n) = \{g \in B_n \mid [g, h] = 1, \text{ for all } h \in B_n\}$ is the subgroup $[[B_n, B_n], B_n]$ generated by all commutators of weight 3. This follows in part from the fact that B_n *is nilpotent of nilpotency class* 3, and thus all commutators of weight 4 are the identity in B_n.

Diameter of B_n We also have the following lemma regarding the diameter of B_n:

[1]Note that $x^{-1} = x^2$ in B_n, since B_n has exponent 3.

Lemma 1. [3] *There exists $\tau_n \in B_n$ such that $\|\tau_n\| \in \Omega(\frac{n^3}{\log n})$, where $\|\cdot\|$ denotes length.*

Proof. Let $d_n = \max_{x \in B_n}(\|x\|)$, and recall that $|B_n| = 3^{n+\binom{n}{2}+\binom{n}{3}}$. Since all elements of the group can be written with at most d_n symbols taken from $x_1^{\pm 1}, \ldots, x_n^{\pm 1}$, we have

$$\begin{aligned}(2n)^{d_n} \geq 3^{n+\binom{n}{2}+\binom{n}{3}} &\iff d_n \log_3(2n) \geq n + \binom{n}{2} + \binom{n}{3} \\ &\iff d_n \geq \left\lceil \frac{n + \binom{n}{2} + \binom{n}{3}}{\log_3 2n} \right\rceil .\end{aligned}$$

This completes the proof. □

2.6 Computational aspects of B_n

In order for the Burnside groups to be of use in cryptography, they must at least satisfy the following: they must have a concise representation, and have efficiently computable group operation. We demonstrate here that both criteria are met.

First, since each element of B_n has unique normal form as a product of the generators and commutators of weights 2 and 3, it is enough to store an array of exponents (where each exponent is either $0, 1, -1$) to represent an element. The size of the array is cubic in n.

As for the group operation, this can be computed simply by concatenating two normal forms, and then reducing the resulting word back into normal form. This process is called the *collection process*. It takes cubic time (see [17, Chapter 11]) in the length of the input.However, since B_n is nilpotent of class 3, all commutators of weight 3 are in the center $Z(B_n)$, and hence there is no need to expand them and apply the collection process: one can simply add the corresponding exponents modulo 3. Furthermore, since all commutators of weight 4 are trivial (see [17, Chapter 18]) we know that $[B_n, B_n]$ is commutative. Hence, we can again avoid the collection process when moving the weight–2 commutators amongst themselves, and in cubic time, we can reduce the expression to a "nearly" normal form con-

sisting of a product of at most $2n$ generators (or their inverses) followed by commutators in normal form. Therefore we need only to apply the collection process on linear input, and so the overall running time of computing the product is indeed $\mathcal{O}(n^3)$.

Inverses can also be computed over B_n in at most cubic time by a similar (yet somewhat simpler) collecting process.

The last and most challenging computational aspect of B_n relates to its *geodesics*: the computation of distances in the Cayley graph. For the applications we introduce here, it will suffice to compute the *norm* (i.e. the distance to the identity of the group).

In general, finding geodesics in the Cayley graph is a difficult problem. In some cases, it is known to be NP–hard [37].[2]

However, this is not as troubling as it seems. We need only to compute norms in the codomain group P_n, which is generally small, and does not necessarily grow with the security parameter (although it may grow with a correctness parameter). For the case of the free Burnside group B_r, one possible solution is to perform a breadth-first search of the Cayley graph, storing the norm of every element in a table. Around $r = 5$, the feasibility of this process becomes questionable.

For the general case, geodesics in the Cayley graph of B_n might be efficiently computable (perhaps up to small approximation factors) making use of a number of commutator identities, for instance:

$$[a, b] = [b, a]^{-1}; a[a, b] = [a, b]a; [a, b]^{-1} = [a^{-1}, b]; [a, b, c]^{-1} = [a^{-1}, b, c].$$

This might allow one to either shorten the normal forms, or to first reorder the generators and then notice that the resulting words are shorter. We shall not concern ourselves with this here, but will consider this problem separately.

[2]One entertaining example is that of the Rubik's cube group, whose diameter was demonstrated to be 20 in 2010 via a distributed computing project which required 35 CPU-years.

To begin with, $|B(n)|$ grows superexponentially with respect to n, so the size of the group is large enough to prevent a brute force attack. Moreover, $B(n)$ is *relatively free*, so every self-map on the generators extends to a unique endomorphism. Perhaps more important to this particular application is the fact that despite the relatively small order of the group, there are group homomorphisms of superpolynomial order. We consider the homomorphism corresponding to the following map of generators $a_1, a_2, \ldots, a_n$:

$$a_i \mapsto a_{i+1}, \quad 1 \leq i < n,$$
$$a_n \mapsto a_1 a_n.$$

We will show experimental results related to this in Section 3.4.

The protocol using Burnside groups works the same as described in Section 2.3.2. The efficiency of the computations (namely the cost of computing exponents) in $B(n)$ is explored in the next chapter.

Chapter 3

Burnside groups as platform for HKKS

In order for a group to be a suitable platform for the HKKS key exchange protocol, we at the very least need to demonstrate that endomorphisms of that group do not have small order. In this chapter, we will describe methods for determining a *cycle* or *loop* of a set map, as well as methods of solving the discrete logarithm problem.

The discrete logarithm problem is of fundamental importance in public-key cryptography: many commonly used cryptosystems are based on the assumption that the discrete logarithm is difficult to compute. In this chapter, we describe some methods for solving the discrete logarithm problem, in particular Pollard's rho method in Section 3.2 and Shank's baby-step, giant-step algorithm in Section 3.3.

Finally, we will describe our experiments performed in GAP [14] in Section 3.4 which show that the Burnside groups of exponent 3 are a good candidate for a platform for the HKKS key exchange protocol.

3.1 Cycle detection

Cycle detection is the problem of finding a cycle in a sequence of iterated function values.

For any finite set S, any initial value $x_0 \in S$ and any function $f\colon S \to S$, the sequence

$$x_0, x_1 = f(x_0), x_2 = f(x_1), \ldots, x_i = f(x_{i-1}), \ldots$$

must eventually use the same value twice.

Given f and $x_0 \in S$, find i, j such that $x_i = x_j$. We will focus on group endomorphisms, $h\colon G \to G$, where G is a finite group.

Let μ be the smallest index such that x_μ appears infinitely often in the sequence $\{x_i\}$, and let λ denote the smallest positive integer such that $x_\mu = x_{\lambda+\mu}$. In other words, μ is the start of the loop and λ is the length of the loop. So we want to find a way to determine μ and λ.

The naïve approach to cycle detection is as follows. Given a group homomorphism $h\colon G \to G$, with G a finite group, and an element $g \in G$, we can store the elements of the sequence $\{h^i(g)\}$ in a list. Then compare each new list element with all the previous ones, looking for a repeat.

The problem with this method is that it is slow and uses a lot of memory. The space complexity is maybe $\lambda + \mu$, which is unnecessarily large. Part of what is inefficient for some groups is computing each power of h.

A technique that is often employed to increase efficiency in computation is the square-and-multiply method. With this method, we precompute powers of h:

$$h, h^2, h^4, h^8, \ldots h^{\lfloor \log_2(|G|) \rfloor}.$$

Then, for each m from the naïve approach, square and multiply. For example, when $m = 17$,

then we write $m = 1 + 16$ and so

$$h^{17} = h \cdot h^{16}.$$

This is now just the composition of two maps, which were already computed.

The main problem with square-and-multiply is that it does not really improve our memory issues. For memory purposes, we would like to find μ and λ while examining as few values from the sequence as possible.

We should develop a way to do this which uses significantly less memory than it would take to store the entire sequence. Applying the equality test to each pair of values results in quadratic time overall. We can do better.

Tortoise and hare

The *tortoise and hare algorithm* is a pointer algorithm, due to Robert W. Floyd, and was developed in the late 1960's [28].

There are two pointers: the tortoise (T) and the hare (H). They move at different speeds through the sequence. We need only check for repeated values of the form

$$x_i = x_{2i}.$$

At each step, i increases by 1, so T moves forward 1 and H moves forward by 2. Then we compare the sequence values at these pointers. The first thing we look for is the smallest value i such that T=H.

Now, the position of T (the distance between T and H) is divisible by the length of the cycle. So, H is moving in the cycle and T (set to g) is moving towards the cycle. They will intersect at the beginning of the cycle. Call this point μ. We find the position of the first repetition of length μ. Now T and H move at the same speed. (Note that H=$h^{2i}(g)$.)

We now have μ, the beginning of the cycle. We now find the length of the shortest cycle from $x_\mu = h^\mu(g)$. T stays still, H moves.

This algorithm only accesses the sequence by storing and copying pointers, functions evaluations and equality tests. It uses $O(\lambda + \mu)$ operations of these types, and $O(1)$ storage space.

3.2 Pollard's ρ method

Pollard's ρ algorithm for logarithms was introduced in 1978 [45]: the goal is to find γ such that $\alpha^\gamma = \beta$, where $\beta \in \langle\alpha\rangle =: G$. Like the tortoise and hare algorithm, it is based on Floyd's cycle-finding algorithm [28]. The algorithm uses Floyd's algorithm to compute a, b, A, B such that $\alpha^a\beta^b = \alpha^A\beta^B$. Note that in the simplified case, if the underlying group is cycle of order n, then γ is a solution to

$$(B - b)\gamma \equiv (a - A) \pmod{n}.$$

To find a, b, A, B, use Floyd's cycle-finding algorithm to find a cycle in the sequence $x_i = \alpha^{a_i}\beta^{b_i}$, where the function $f\colon x_i \mapsto x_{i+1}$ is assumed to be random-looking, and thus enter into a loop after approximately $\sqrt{(\pi n)/2}$ steps. One way to define such a function is to divide G into three disjoint subsets of approximately equal size: S_0, S_1, S_2. If x is in S_0, then double both a and b; if x is in S_1, increment a; if x is in S_2, increment b.

Algorithm 3. Let G be a cyclic group of order p, let $a, b \in G$ be given, let $G = S_0 \cup S_1 \cup S_2$

be a disjoint partition, and let $f\colon\ G \to G$ be a map

$$f(x) = \begin{cases} \beta x & x \in S_0, \\ x^2 & x \in S_1, \\ \alpha x & x \in S_2. \end{cases}$$

Then define maps $g, h\colon\ G \times \mathbb{Z} \to \mathbb{Z}$ by

$$g(x,n) = \begin{cases} n & x \in S_0, \\ 2n \pmod p & x \in S_1, \\ n+1 \pmod p & x \in S_2, \end{cases}$$

$$h(x,n) = \begin{cases} n+1 \pmod p & x \in S_0, \\ 2n \pmod p & x \in S_1, \\ n & x \in S_2. \end{cases}$$

The algorithm takes as input a generator a of G and an element b of G. The output is an integer x such that $a^x = b$, or failure. The following pseudocode describes how to find x:

(1) Initialize

$$a_0 \leftarrow 0,$$

$$b_0 \leftarrow 0,$$

$$x_0 \leftarrow 1 \in G,$$

$$i \leftarrow 1.$$

(2) Define

$$\begin{aligned} x_i &\leftarrow f(x_{i-1}), & & x_{2i} \leftarrow f(f(x_{2i-2})), \\ a_i &\leftarrow g(x_{i-1}, a_{i-1}), & \text{and} \quad & a_{2i} \leftarrow g(f(x_{2i-2}, g(x_{2i-2}), a_{2i-2})), \\ b_i &\leftarrow H(x_{i-1}, a_{i-1}), & & b_{2i} \leftarrow h(f(x_{2i-2}, h(x_{2i-2}), a_{2i-2})) \end{aligned}$$

(3) If $x_i = x_{2i}$, then

(a) $r \leftarrow b_i - b_{2i}$,

(b) if $r = 0$, return 'failure',

(c) $x \leftarrow r^{-1}(a_{2i} - a_i) \pmod{p}$,

(d) return x.

(4) If $x_i \neq x_{2i}$, then $i = i + 1$ and return to step 2.

The running time is approximately $O(\sqrt{p})$, where p is n's largest prime factor.

3.3 Baby-step, giant-step

Shank's *baby-step, giant-step* algorithm is a method for computing the discrete logarithm of an integer h not divisible by p with respect to a primitive root g modulo p. When employed correctly, it is significantly faster than the brute-force method of checking all $p - 1$ powers of g modulo p and checking which power matches h.

The idea is this: let $m = \lceil \sqrt{p} \rceil$. If the discrete logarithm of h is x, then we can write $x = mq + r$ for some integers q, r such that $0 \leq r < m$. Then

$$h \equiv g^x = g^{mq} g^r \pmod{p},$$

which means that $h(g^{-m})^q \equiv g^r \pmod{p}$. The strategy then is to compute the m numbers $1, g, g^2, \ldots, g^{m-1}$ modulo p, and then compute g^{-m} modulo p. Then we start computing

$h(g^{-m})^q$ modulo p for $q = 0, 1, 2, \ldots$. At some point, we will have a collision with the first list produced, i.e. we will find some q such that $h(g^{-m})^q \equiv g^r \pmod{p}$ for some $0 \leq r < m$. Once this collision is found, the discrete logarithm we are looking for is $mq + r$.

Example. Let $p = 31$ and $g = 3$ (note that 3 is a primitive root modulo 31). Let us compute the discrete logarithm of $h = 6$.

We have $m = \lceil \sqrt{31} \rceil = 6$, and compute $1, g, g^2, \ldots, g^5$ modulo 31:

$$g^0 = 1, \quad g^1 = 3, \quad g^2 = 9, \quad g^3 = 27, \quad g^4 = 19, \quad g^5 = 26.$$

We also compute $g^{-m} = 3^{-6} \pmod{31} \equiv 21^6 \equiv 2 \pmod{31}$, and then start computing $h(g^{-m})^q = 6 \cdot 2^q$ for increasing values of q:

$$\begin{aligned} h(g^{-m})^0 &= 6 \cdot 2^0 = 6, \\ h(g^{-m})^1 &= 6 \cdot 2^1 = 12, \\ h(g^{-m})^2 &= 6 \cdot 2^2 = 24, \\ h(g^{-m})^3 &= 6 \cdot 2^3 \equiv 17, \\ h(g^{-m})^4 &= 6 \cdot 2^4 \equiv 3, \end{aligned}$$

and we've found a collision: $h(g^{-m})^4 \equiv g^1 \pmod{p}$. This means that

$$h \equiv g^{4m+1} = g^{25},$$

so 25 is the discrete logarithm we seek.

Let's consider the number of operations this algorithm requires. To begin with, we need to compute m different powers of g, which means m multiplications. To compute g^{-m} modulo p, after we invert, we can use the square-and-multiply method to raise g^{-1} to the m^{th} power in

$\log(m)$ steps. Then we need to compute $h(g^{-m})^q$ for various values of q. Since $x = mq + r$, we have $q = \frac{x-r}{m} \leq \frac{x}{m} \leq \frac{p}{\sqrt{p}} = \sqrt{p} \leq m$, so at worst we will need to compute up to $q = m$. But for each of these, in addition to the multiplication, we also have to search the original list of m elements. We can use a binary search to do this in $\log(m)$ comparisons. So at worst, we need to do $m\log(m)$ multiplications and comparisons as we compute $h(g^{-m})^q$ to look for collisions. So we end up with at worst $m + 1 + \log(m) + m\log(m)$ multiplications, inversions and comparisons, which is $O(m\log(m))$. Since m is $O(\sqrt{p})$, then $m\log(m)$ is $O(\sqrt{p}\log(\sqrt{p})) = O(\sqrt{p}\log(p))$. This is obviously faster than the naïve brute-force approach, which is $O(p)$.

3.4 Experimental results

Recall that since we are only considering Burnside groups of exponent 3, we use the notation B_n where n is the number of generators. Recall also that a set map on the generators of B_n extends uniquely to an endomorphism, so we can fully describe a homomorphism on B_n by how it maps the generators $x_1, x_2, \ldots, x_n$.

We began by computing the order of all homomorphisms of B_n for small values of n, and discovered that the map ϕ_n given by

$$x_i \mapsto x_{i+1} \quad 1 \leq i < n, \quad x_n \mapsto x_1 x_n,$$

gave a homomorphism of a large order. Note that here we are using the term *order* rather loosely, as what we really mean is *cycle*, i.e. find i, j such that $\phi_n^i = \phi_n^j$. If ϕ_n is an automorphism then it is the special case of cycle detection with $i = 1$.

We soon realized that the order of the above-described map grows very quickly as n grows. Enlisting a supercomputer through the CUNY High Performance Computing Cen-

ter[1] computing the order of these homomorphisms using GAP was inefficient, even with parallelization. For ease of computation, we shifted our focus to the abelianization of B_n, that is, B_n "modded out by the commutators:" this allowed us to work with $n \times n$ matrices over $\mathbb{Z}_3$. Then any homomorphism ϕ over B_n corresponds to one in this abelianization. If we can show a lower bound "big enough" here, then it extends back to B_n, since the order of the matrix representation of ϕ will divide the order of the homomorphism ϕ. For our particular map ϕ_n, let M_n denote the corresponding $n \times n$ matrix.

We computed that the order of the matrix corresponding to ϕ_{22}, which is denoted by M_{22}, is $15,625,959,602$, and the order of the matrix corresponding to ϕ_{24}, denoted M_{24}, is $10,460,353,202$. Putting these together in a 46×46 block matrix

$$M := \begin{pmatrix} M_{22} & 0 \\ 0 & M_{24} \end{pmatrix},$$

we see that $|M| = \text{lcm}(M_{22}, M_{24}) > 2^{65}$, where $|M| \mid |\phi_{46}$.

These computations were done with an Intel Core i5 dual-core 2.5 GHz processor and 4 GB of RAM running Ubuntu Linux version 14.10, running GAP version 4.7.5.

As far as the computation time, we averaged about 0.081 seconds per multiplication, without any optimization and using GAP. This is slower than ideal. Moreover, as we are working with matrices, the linear algebra attacks described in Section 2.4 apply here. However, if we instead use ϕ_{46} as the homomorphism for the semidirect product in the protocol, then known attacks do not work.

[1]The CUNY HPCC is operated by the College of Staten Island and funded, in part, by grants from the City of New York, State of New York, CUNY Research Foundation, and National Science Foundation Grants CNS-0958379, CNS-0855217 and ACI 1126113.

Chapter 4

Hashing with Matrices

Along with key exchange, a very important cryptographic primitive is the *hash function*. Cryptographic hash functions have many applications to information security, including digital signatures and methods of authentication. They can also be used as ordinary hash functions, to index data in a hash table, fingerprinting (a procedure which maps an arbitrary large data item to a shorter bitstring, or fingerprint which uniquely identifies the original data for all practical purposes), to detect duplicate data and as checksums to detect (accidental) data corruption. In fact, in the context of information security, cryptographic hash values are often referred to as fingerprints, checksums, or just hash values.

In this chapter we will first define the hash function and explain some properties we require a hash function to possess. Then we introduce *Cayley hash functions*, which are a family of hash functions based on nonabelian groups. We then explore hashing with matrices, and propose using matrices over $\mathrm{SL}_2(\mathbb{F}_p)$ of a particular form which generate $\mathbb{F}_p$.

4.1 Background

In this section we introduce the definition and the main properties of cryptographic hash functions. Hash functions are fundamental to cryptographic protocols, particularly useful in digital signature schemes and message authentication. A *hash function* is a function that maps a bitstring of an arbitrarily long length to a fixed (small) length bitstring. Additionally, we require the function to be collision resistant, second preimage resistant and preimage resistant.

Definition 6. Let $n \in \mathbb{N}$ and let $H\colon \{0,1\}^* \to \{0,1\}^n$ such that $m \mapsto h = H(m)$. We require a hash function to satisfy the following:

(1) *preimage resistance*: given output y, it is hard to find input x such that $H(x) = y$;

(2) *second preimage resistance*: given input x_1, it is hard to find another input $x_2 \neq x_1$ such that $H(x_1) = H(x_2)$;

(3) *collision resistance*: it is hard to find inputs $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$.

Note that since hash functions are not injective, this 'uniqueness' that we desire is purely computational. From a practical perspective, this means that no big cluster of computers can find the input based only on the output of a hash function.

There exist old hash function constructions whose collision resistance follows from the hardness of number-theoretic or group-theoretic problems. However, these hash functions can only be used in applications which require only collision resistance and are often too slow for practical purposes. Standardized hash functions, such as the SHA family, follow the *block cipher design*: their use is not restricted to collision resistance, but their collision resistance is heuristic and not established by any precise mathematical problem. In fact, recent attacks against the SHA-1 algorithm have led to a competition for a new Standard hash Algorithm [40].

Another direction, more relevant to our work, is the *expander hash function*, dating back to 1991 when Zémor proposed building a hash function based on the special linear group. This first attempt was quickly broken, but Tillich and Zémor quickly proposed a second function which was resistant to the attack on the first; see [52]. However, as we will describe, this newer hash function is also vulnerable to attack; see Section 4.2.1. The Tillich–Zémor hash function is a type of expander hash called a *Cayley hash function*, and is different from functions in the SHA family in that it is not a block hash function, but rather each bit is hashed individually. We discuss this particular hash function in further detail in Section 4.2.

The expander hash design is fundamentally different from classical hash designs in that it allows for relating important properties of hash functions such as collision resistance, preimage resistance (see Definition 6) and their output distribution to the graph-theoretical notions of cycle, girth and expanding constants. When the graphs used are *Cayley graphs*, the design additionally provides efficient parallel computation and group-theoretical interpretations of the hash properties.

The expander hash design, though not so new anymore, is still little understood by the cryptographic community. The Tillich–Zémor hash function is often considered broken because of existing trapdoor attacks and attacks against specific parameters. In fact, relations between hash, graph and group-theoretic properties have been sketched but no precise statements on these problems exist. Since the mathematical problems which underly the security of expander hashes do not belong to classical problems, it appears as though they have not been investigated. Hence their actual hardness is unknown. Efficiency aspects have also only been sketched.

The goal of the work on hash functions in this thesis is to establish new (Cayley) hash functions which are resistant to known attacks, and demonstrate their computational efficiency.

Cayley hash functions are based on the idea of using a pair of (semi)group elements, A

and B, to hash the 0 and 1 bit, respectively, and then to hash an arbitrary bitstring by using multiplication of elements in the (semi)group. We focus on hashing with 2×2 matrices over $\mathbb{F}_p$. Since there are many known pairs of 2×2 matrices over $\mathbb{Z}$ which generate a free monoid, this yields numerous pairs of matrices over $\mathbb{F}_p$ (for p sufficiently large) that are candidates for collision-resistant hashing. However, this trick can backfire and allow for a lifting of matrix elements to $\mathbb{Z}$ to find a collision. This "lifting attack" was used by Tillich and Zémor [52] in the case where two matrices A and B generate (as a monoid) all of $\mathrm{SL}_2(\mathbb{Z}_+)$. We will show that with other, 'similar' pairs of matrices from $\mathrm{SL}_2(\mathbb{Z})$, the situation is different, and while the same "lifting attack" can (in some cases) produce collision in the *group* generated by A and B, it says nothing about the *monoid* generated by A and B. Since we only use positive powers for hashing, this is all we need, and we argue that for these pairs of matrices, there are no known attacks at this time that would affect the security of the corresponding hash functions.

Additionally, we give lower bounds on the length of collisions for hash functions corresponding to some particular pairs of matrices from $\mathrm{SL}_2(\mathbb{F}_p)$.

4.1.1 Cayley hash functions

Classical hash functions mix pieces of the message repeatedly so the result appears sufficiently random [44]. For this reason, they may be unappealing outside the area of cryptography. On the other hand, a particular type of expander hash function, the Cayley hash function, has a more straightforward design.

Given a group G and a subset $S = \{s_1, \ldots, s_k\}$ of G, their *Cayley graph* $\mathfrak{G}$ is a k–regular graph that has a vertex v_g associated to each element of G and an edge between vertices v_{g_1} and v_{g_2} if there exists $s_i \in S$ such that $g_2 = g_1 s_i$.

To build a hash function from the Cayley graph, let $\sigma\colon \{1, \ldots, k\} \to S$ be an ordering, fix an initial value g_0 and write the message m as a string $m_1 m_2 \cdots m_N$, where $m_i \in \{1, \ldots, k\}$.

Then the hash value is $H(m) := g_0\sigma(m_1)\cdots\sigma(m_N)$. This is represented on the Cayley graph as a (nonbacktracking) walk; the endpoint of the walk is the hash value.

Two texts yielding the same hash value correspond to two paths with the same start and endpoints. We would like those two paths to differ necessarily by a "minimum amount". Such a vague notion can be guaranteed if there are no short cycles in the Cayley graph. More precisely, we want the Cayley graph to have a large *girth*:

Definition 7. The *directed girth* of a Cayley graph $\mathfrak{G}$ is the largest integer ∂ such that, given any two vertices u and v, any pair of distinct paths which joins u to v will be such that one of those paths has length (i.e. number of edges) ∂ or more.

The idea is that the girth of the Cayley graph is a relevant parameter to hashing. More precisely, if a Cayley graph has a large girth ∂, then the corresponding hash function will have the property that small modifications of the text will modify the hash value [52].

One of the main advantages of Cayley hash functions over classical hash functions is their ability to be parallelized. Namely, if messages x and y are concatenated, then the hashed value of xy is $H(xy) = H(x)H(y)$. Associativity of the group means we can break down a large message into more manageable pieces, hash each piece, and then recover the final result from the partial products.

Finally, a desirable feature of any hash function is the equidistribution of the hashed values. This property can be guaranteed if the associated Cayley graph satisfies the following property.

Proposition 1. [53, Proposition 2.3] *If the Cayley graph of a group G is such that the greatest common divisor of its cycle lengths equals* 1*, then for the corresponding hash function, the distribution of hashed values of texts of length n tends to equidistribution when n tends to infinity.*

This proposition is proved using classical graph-theoretic techniques, by studying successive powers A^n of the adjacency matrix of the graph. Equidistribution can be achieved with graphs that have a high expansion coefficient (see [54]).

The collision, second preimage and preimage resistance of classical hash functions easily translates to group-theoretic problems.

Definition 8. Let G be a group and let $S = \{s_1, \ldots, s_k\} \subset G$ be a generating set. Let $L \in \mathbb{Z}$ be 'small.'

(1) *Balance problem* Find an "efficient" algorithm that returns two words $m = m_1 \cdots m_\ell$ and $m' = m'_1 \cdots m'_{\ell'}$ with $\ell, \ell' < L$, $m_i, m'_i \in \{1, \ldots, k\}$ and $\prod s_{m_i} = \prod s_{m'_i}$.

(2) *Representation problem* Find an "efficient" algorithm that returns a word $m_1 \cdots m_\ell$ with $\ell < L$, $m_i \in \{1, \ldots, k\}$ and $\prod s_{m_i} = 1$.

(3) *Factorization problem* Find an "efficient" algorithm that, given any element $g \in G$, returns a word $m_1 \cdots m_\ell$ with $\ell < L$, $m_i \in \{1, \ldots, k\}$ and $\prod s_{m_i} = g$.

Note that since the group is finite, the length restriction is required, since for every $w \in G$, $w^{|G|} = 1$. Note also that Lubotzky described the factorization problem as a noncommutative analog of the discrete logarithm problem [33]. In fact, if we omit trivial solutions, then the representation and factorization problems are equivalent to the discrete logarithm problem in abelian groups.

In general, the factorization problem is at least as hard as the representation problem, which is itself at least as hard as the balance problem.

It is apparent that a Cayley hash function is collision resistant if and only if the balance problem is hard, second preimage resistant if and only if the representation problem is hard, and preimage resistant if and only if the factorization problem is hard.

Among all Cayley hash proposals, the Tillich–Zémor hash function is the only remaining current candidate. (We will propose later in this chapter a similar hash function which is

resistant to known attacks on parameters of the Tillich–Zémor hash function.) In general, the security of Cayley hashes depends on the hardness in general of the factorization problem, which remains a big open problem.

The efficiency of Cayley hashes depends on specific parameters: the Tillich–Zémor hash function is the most efficient expander hash, but it is still 10 to 20 times slower than the standard classical hash SHA. Computation in Cayley hashes can be easily parallelized, which could be a major benefit in applications. In this chapter we propose a hash function based on the Tillich–Zémor hash function which is resistant to known methods of attack, and which is efficient in computation.

4.1.2 Possible attacks

The mathematical structure of Cayley hash functions leaves them vulnerable to attacks which exploit this structure.

An important category of attack is the *subgroup attack*. A probabilistic attack was devised by Camion [6], based on the search for text whose hashcode falls into a subgroup.

A second important category of attack is the *lifting attack*. Let us illustrate how a lifting attack works with an example. Let $G = \mathrm{SL}(2, \mathbb{F}_p)$. There is a natural map, the *reduction modulo p map*, from $\mathrm{SL}(2, \mathbb{Z})$ to $\mathrm{SL}(2, \mathbb{F}_p)$. A lifting attack for $\mathrm{SL}(2, \mathbb{F}_p)$ will 'lift' the generators of $\mathrm{SL}(2, \mathbb{Z})$ and then try to 'lift' the element to be factored on the subgroup of $\mathrm{SL}(2, \mathbb{Z})$ generated by the lifts of the generators. Generally, if a factorization exists, it is easier to find over $\mathbb{Z}$ rather than over $\mathbb{F}_p$, since properly chosen generators of an infinite group will give us unique factorization. Once a factorization over $\mathbb{Z}$ has been obtained, reducing modulo p provides a factorization over $\mathbb{F}_p$. The most difficult part of the lifting attack is the lifting itself.

We will explore later how this lifting works in particular cases. In particular we will examine the first Cayley hash function proposed by Tillich and Zémor [52]. They chose

$G = \mathrm{SL}(2, F_p)$ with generating set

$$S = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right\}.$$

The problem is that this group actually generates all of $\mathrm{SL}(2, \mathbb{Z}^+)$. Using the Euclidean algorithm, factorizations in this infinite group are computed easily. We will propose a simple modification of the generators which counters the lifting attack.

4.2 The Tillich–Zémor hash function

Hashing with matrices refers to the idea of using a pair of matrices, A and B (over a finite ring) to hash the "0" bit and the "1" bit, respectively. Then, an arbitrary bit string is hashed by using multiplication of matrices. So, the bit string 1001101 is hashed to the matrix BA^2B^2AB.

One way to help ensure the requirements of Definition 6 is to use a pair of elements, A and B, of a semigroup S such that the Cayley graph of the semigroup generated by A and B is an expander graph. The most popular implementation of this idea is the *Tillich–Zémor hash function* [53].

The use of the special linear group $\mathrm{SL}_2(\mathbb{F}_p)$ of 2×2 matrices with determinant 1 over a finite field $\mathbb{F}_p$ is a promising choice for devising hash functions. To begin with, we can choose simple matrices as generators, which yields a fast hash: multiplication by such a matrix amounts to a few additions in $\mathbb{F}_p$. Cayley graphs over $\mathrm{SL}_2(\mathbb{F}_p)$ also have good expanding properties; see Sarnak [48], Lafferty and Rockmore [31], and Margulis [34, 35].

4.2.1 An efficient attack on the Tillich–Zémor hash function

In this section we will outline the attack described in [52]. For the hash function defined using the above matrices $A(1)$ and $B(1)$, we can obtain collisions by looking for short factorizations of the identity into a product of $A(1)$'s and $B(1)$'s in $\mathrm{SL}(2, \mathbb{F}_p)$. Note the following.

Proposition 2 ([52])**.** *The matrices*

$$A(1) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \textit{and} \quad B(1) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

freely generate the monoid

$$\mathbf{M} = \left\{ M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \;\middle|\; a, b, c, d \in \mathbb{N},\ ad - bd = 1 \right\}.$$

Moreover, there is an explicit algorithm which factorizes any matrix of $\mathbf{M}$ *into a product of* $A(1)$*'s and* $B(1)$*'s and runs in a logarithmic number of steps in* $\max(a, b, c, d)$*.*

In fact, the algorithm alluded to in the above proposition is the euclidean algorithm applied to the pair (a, b) if $a + b > c + d$, and to (c, d) otherwise. In fact, if $a + b > c + d$, $a > b$ and the number of steps for the euclidean algorithm to terminate, n, is even, then we have the factorization

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = A(1)^{q_n} B(1)^{q_{n-1}} \cdots A(1)^{q_2} B(1)^{q_1},$$

where q_i is the i^{th} quotient that appears in the application of the algorithm.

Similarly, if $c + d > a + b$ and the number of steps it takes for the euclidean algorithm to

terminate, n, is odd, then we have the factorization

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = B(1)^{q_n} A(1)^{q_{n-1}} \cdots A(1)^{q_2} B(1)^{q_1},$$

where the q_i are the quotients that appear in the algorithm.

Now we have the following strategy for an attack. First, find a matrix of the form

$$M = \begin{pmatrix} 1 + k_1 p & k_2 p \\ k_3 p & 1 + k_4 p \end{pmatrix}$$

in $\mathrm{SL}(2, \mathbb{Z})$, with nonnegative coefficients, which reduces to the identity modulo p. Then use the euclidean algorithm (on (a, b) if $a + b > c + d$ and on (c, d) otherwise) to factor M into a product of $A(1)$'s and $B(1)$'s over $\mathrm{SL}(2, \mathbb{Z})$. Reducing modulo p, this leads to a factorization of the identity in $\mathrm{SL}(2, \mathbb{F}_p)$.

The first question to answer is whether there exists such an M which is nontrivial. To this end, we must find four integers k_1, k_2, k_3, k_4, not all 0, which correspond to the matrix M having determinant 1. In other words, we must solve the diophantine equation

$$(1 + k_1 p)(1 + k_4 p) - k_2 p k_3 p = 1. \tag{4.1}$$

To avoid the factorization being too long, we want all k_i of roughly the same order of magnitude. To do this, we outline the algorithm described in [52].

Algorithm 4. Choose a small integer c. Search for a solution (k_1, k_2, k_3, k_4) such that $k_1 + k_4 = cp$. To obtain this, choose a random prime p' of roughly the same magnitude as p, and take $k_3 = p'$. Now (4.1) becomes

$$k_2 p' = c + c p k_1 - k_1^2. \tag{4.2}$$

This is possible if we can solve $k_1^2 - cpk_1 - c = 0 \pmod{p'}$, i.e., the discriminant $c^2p^2 + 4c$ must be a quadratic residue modulo p'. With high probability, such a pair (c, p') will be found, giving us the following solution:

$$\begin{aligned} k_1 &= \frac{cp + \sqrt{c^2p^2 + 4c}}{2}, \\ k_2 &= \frac{c + cpk_1 - k_1^2}{p'}, \\ k_3 &= p', \\ k_4 &= cp - k_1. \end{aligned}$$

In k_1, we take the square root modulo p'.

These solutions (k_1, k_2, k_3, k_4) are generally of order $O(p)$, which yields a matrix M of order $O(p^2)$.

Let us now illustrate how the algorithm works with a simple example.

Example. Let $p = 7$ and take $c = 1$. Then $c^2p^2 + 4c = 53$, which is a quadratic residue modulo $11 =: p'$. Then, using the above formula for finding the integers k_1, k_2, k_3, k_4, we get

$$k_1 = 5, \quad k_2 = 1, \quad k_3 = 11, \quad k_4 = 2.$$

This corresponds to the matrix

$$M = \begin{pmatrix} 36 & 7 \\ 77 & 15 \end{pmatrix}.$$

Applying the euclidean algorithm to the pair $(77, 15)$, we get quotients $q_1 = 5, q_2 = 7, q_3 = 2$. We can factor M as a product of $A(1)$'s and $B(1)$'s as

$$M = \begin{pmatrix} 36 & 7 \\ 77 & 15 \end{pmatrix} = B^2A^7B^5 \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \pmod{7}.$$

This lifting attack allows for collision in the hash function, thus violating one of the conditions we require.

4.3 Hashing with $G = \mathrm{SL}(2, \mathbb{F}_p)$

Another idea is to use A and B over $\mathbb{Z}$ which generate a free monoid, and then reduce the entries modulo a large prime p to get matrices over $\mathbb{F}_p$. Here we have a lower bound on the length of bit string where a collision may occur, since there cannot be an equality of positive products of A and B unless at least one of the entries in at least one of the products is at least p. We will show precisely what this bound is, but it is on the order of $\log p$.

In this section we investigate the Cayley graphs of $\mathrm{SL}(2, \mathbb{F}_p)$ generated by

$$A(n) = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}, \quad B(n) = \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix},$$

where $n = 2, 3$, respectively, and p is a large prime. Particularly, we show their application to hashing.

The main difference is the difference between the *group* generated by $A(n)$ and $B(n)$ and the *monoid* generated by $A(n)$ and $B(n)$.

4.3.1 The base case

A pair of matrices over $\mathbb{Z}$ which generate a free monoid is

$$A(1) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \quad B(1) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Note that these matrices as generators of the group $\mathrm{SL}(2, \mathbb{F}_p)$ have a Cayley graph which forms an expander, so they are good candidates for the basis of a hash function. Note also that these matrices are invertible and thus actually generate the group $\mathrm{SL}(2, \mathbb{Z})$. This group is not free, but the monoid generated by $A(1)$ and $B(1)$ is free. Since only positive powers are used in hashing, this is all we need.

However, since $A(1)$ and $B(1)$ generate all of $\mathrm{SL}(2, \mathbb{Z})$, we can use a lifting attack on the corresponding hash function: a collision is found using the Euclidean algorithm on the entries of a matrix; see Section 4.2.1. In short, it is readily seen that a short factorization of the identity over $\mathrm{SL}(2, \mathbb{F}_p)$ produces collisions. To find such a factorization, the strategy is to reduce the problem to factoring in an infinite group; in this case, the group $\mathrm{SL}(2, \mathbb{Z})$. Find a matrix U in $\mathrm{SL}(2, \mathbb{Z})$ which reduces modulo p to the identity, and which can be expressed as a product of $A(1)$'s and $B(1)$'s. In this case, that means that we only require U to have nonnegative coefficients. Then we use the Euclidean algorithm, which is an efficient way to obtain the factorization of U.

For this attack to be effective, there must be a way of finding such a matrix U. Tillich and Zémor [52] describe a probabilistic algortihm which does this. It is based on the fact that the set of matrices of $\mathrm{SL}(2, \mathbb{Z})$ with nonnegative coefficients is 'dense.'

To protect against such attacks, one should choose a set of generators that generate a sufficiently sparse submonoid of the infinite group associated with $\mathrm{SL}(2, \mathbb{F}_p)$. Tillich and Zémor proposed using the matrices

$$A = \begin{pmatrix} \alpha & 1 \\ 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} \alpha & \alpha + 1 \\ 1 & 1 \end{pmatrix},$$

where computations are made in the quotient field $\mathbb{F}_{2^n} = \mathbb{F}_2/\langle p(x) \rangle$, where $p(x)$ has degree n, and α is a root of p. See [53] for details on the imnplementation of this hash.

Tillich and Zémor use matrices A, B from the group $\mathrm{SL}(2, R)$, where R is a commutative ring defined by $R = \mathbb{F}_2[x]/(p(x))$. They took $p(x)$ to be the irreducible polynomial $x^{131} + x^7 + x^6 + x^5 + x^4 + x + 1$ over $\mathbb{F}_2[x]$. Thus, R is isomorphic to $\mathbb{F}_{2^n}$, where n is the degree of the irreducible polynomial $p(x)$. Then, the matrices A and B are

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

4.3.2 Efficient attacks

This hash function was published in 1994 [53], but there have been several recent attacks. In 2010, Petit and Quisquater [42] describe a preimage attack; in 2011, Grassl, Ilic, Magliveras and Steinwandt [15] describe a collision attack.

Collision attack

In 2011, Grassl, Ilić, Magliveras and Steinwandt [15] provided a method describing a way to obtain collision. They develop some results on collisions between palindromic bitstrings and combine these with a result of Mesirov and Sweet. This reduces the identification of collisions for the Tillich–Zémor hash function to a linear algebra problem which can be easily solved using a computer algebra system.

First, they show that the search for collisions for the Tillich–Zémor hash function can be transferred to the search for collisions when new, specific generators $c_0 := A$ and $c_1 := A^{-1}BA$ are used instead of A and B. In other words, finding collisions for the Tillich–Zémor scheme is equivalent to finding collisions for the hash function h corresponding to C_0 and C_1.

Then, working inside the group $\mathrm{SL}_2(\mathbb{F}_2[x])$ of unimodular matrices over the polynomial ring $\mathbb{F}_2[x]$, define matrices $C_0 = \left(\begin{smallmatrix} x & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and $C_1 = \left(\begin{smallmatrix} x+1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, and denote the corresponding hash function by H. They then apply H to the subset of palindromic bitstrings, and conclude

that if v is a palindromic bitstring, and $H(v) = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, then $b = c$, $\deg(a) = |v|$ and $\max\{\deg(b), \deg(d)\} \leq |v|$. (The proof is by induction on the length $|v|$ of v.)

Next, examine the function ρ given by $\rho(v) = H(0v0) + H(1v1)$. We are interested in evaluating ρ modulo a given irreducible polynomial, because we $\rho(v) \equiv \left(\begin{smallmatrix} 0 & 0 \\ 0 & 0 \end{smallmatrix}\right) \pmod{q(x)}$ if and only if $h(0v0) = h(1v1)$ is in fact a collision in $\mathrm{SL}_2(\mathbb{F}_2[x]/(q(x)))$. Notice that if v is a palindrome of length $|v|$, then $\rho(v) = \left(\begin{smallmatrix} a & a \\ a & 0 \end{smallmatrix}\right)$, where $a \in \mathbb{F}_2[x]$ has degree $|v|$, and that moreover, a is the upper left entry of $H(v)$.

For their attack, they care only about palindromes of even length, i.e. ones that can be written in the form vv^r for some bitstring v (v^r denotes the "reversal" of v). In the case of palindromes of even length, we have that $H(v) = \left(\begin{smallmatrix} a^2 & b \\ b & d^2 \end{smallmatrix}\right)$ for some $a, b, d \in \mathbb{F}_2[x]$.

Combining these results, if v is a palindrome of even length, then $\rho(v) = \left(\begin{smallmatrix} a^2 & a^2 \\ a^2 & 0 \end{smallmatrix}\right)$ for some $a \in \mathbb{F}_2[x]$ with $\deg(a) = \frac{|v|}{2}$. (Note that a^2 is the upper left entry of $H(v)$.)

Using the above facts, we can derive the recurrence relation that for a given palindrome $v := b_n \cdots b_1 b_1 \cdots b_n$ of length $2n$, for $0 \leq i \leq n$, the square root p_i of the upper left entry of $H(v)$ is given by

$$p_i \begin{cases} 1 & i = 0, \\ x + b_1 + 1 & i = 1, \\ (x + b_i)p_{i-1} + p_{i-2} & 1 < i \leq n. \end{cases}$$

Now, for the given irreducible polynomial $q(x) \in \mathbb{F}_2[x]$ of degree n, we seek a palindrome v of length $2n$ such that $\rho(v) = H(0v0) + H(1v1)$ is the zero matrix in $\mathrm{SL}_2(\mathbb{F}_2[x]/(q(x)))$. To do this, determine a second polynomial $p(x) \in \mathbb{F}_2[x]$ of degree $n-1$ such that $\gcd(q(x), p(x)) = 1$, and such that during the execution of the euclidean algorithm on $(q(x), p(x))$, the successive quotients are all of degree 1, and the degree of each remainder is only one less than the degree of the respective divisor. This will ensure we have a "euclidean algorithm chain" of maximal length. The existence of such a polynomial $p(x)$ is due to a result of Mesirov and

Sweet.

Once a polynomial $p(x)$ has been found, the linear quotients $x + \beta_i$, $1 \leq i \leq n$, ocurring in the euclidean algorithm allow us to derive the bits b_i in the above recurrence relation. Notice that $p_1 = x + b_1 + 1$ and $b_= \beta_i$ for $i > 1$, i.e. β_i needs to be inverted; we denote this inversion by $\hat{\beta}_i$. We therefore obtain the desired collision

$$h(0\beta_n \cdots \hat{\beta}_1 \hat{\beta}_1 \cdots \beta_n 0) = h(1\beta_n \cdots \hat{\beta}_1 \hat{beta}_1 \cdots \beta_n 1).$$

Note that once we have a palindrome collision, we immediately get a different (nonpalindromic) collision with bitstrings of the same length, since for a palindrome v, we have that $h(0v0) = h(1v1)$ if and only if $h(0v1) = h(1v0)$.

For the parameters given by Tillich and Zémor, that is, using the platform group

$$\mathrm{SL}_2(\mathbb{F}_2[x]/(x^{127} + x + 1)),$$

they obtained two bitstrings v_1, v_2 of length n with $h(0v_i v_i^r 0) = h(1v_1 v_1^r 1)$, i.e. two collisions of bitstrings of length $2n + 2$ are obtained.

Preimage attack

Working on results introduced by Grassl et al in 2009, Petit and Quiaquater presented two algorithms for computing preimages of the Tillich–Zémor hash function. First, they show that a small modification of Grassl et al's algorithm provides a second preimage algorithm. Then they reduce the problem of finding preimages to any hash value to the problem of precomputing preimages to a few hash values with certain characteristics; see [42, Section 4] for details.

Finally, they provide two algorithms for the precomputing part, each with their own

advantages depending on the user's approach. The first algorithm produces shorter preimages and is therefore more interesting from a practical perspective. The second algorithm is deterministic, and faster than the first algorithm. Theoretically it may also be more interesting, as there is a proof of success in deterministic cubic time for a(n important) subset of parameters. The authors point out that although the Tillich–Zémor hash function has been successfully broken, other hash functions based on a similar design could remain secure to these attacks.

4.3.3 Hashing with $A(2)$ and $B(2)$

In this section, we discuss circuits in the Cayley graph of $\mathrm{SL}(2, \mathbb{F}_p)$ with generating set $A(2), B(2)$. Note that these matrices also correspond to a Cayley graph which forms an expander graph.

We begin by noting that the lifting attack on the hash function depending on $A(1)$ and $B(1)$ described above is the only published attack on that hash function. We will show that this particular attack does not work with $A(2), B(2)$. In particular, this gives evidence of the security of using these matrices for hashing over $\mathbb{F}_p$ for a large prime p.

First we need to justify why these matrices are better candidates than $A(1)$ and $B(1)$. Recall that when considered as matrices over $\mathbb{Z}$, $A(1)$ and $B(1)$ generate (as a monoid) the entire monoid of 2×2 matrices over $\mathbb{Z}$ with positive entries, $\mathrm{SL}(2, \mathbb{Z}_+)$.

However, this is not the case with $A(2)$ and $B(2)$. We combine the following results of Sanov [47] into a single theorem.

Theorem 1. (1) *The group generated by*

$$A(2) = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}, \quad B(2) = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix},$$

is a free group.

(2) *The subgroup of* $\mathrm{SL}(2,\mathbb{Z})$ *generated by* $A(2)$ *and* $B(2)$ *consists of all invertible matrices of the form*

$$\begin{pmatrix} 1+4m_1 & 2m_2 \\ 2m_3 & 1+m_4 \end{pmatrix}, \qquad (*)$$

where the m_i *are integers.*

Proof. (1) We will construct an action which satisfies the conditions of the *Ping-pong Lemma*, which says that if G acts on a set X, and $a, b \in G$, $C, D \subseteq X$ are such that neither C nor D are contained in the other, and further that $b^n(C) \subseteq D$ and $a^n(D) \subseteq C$ for all nonzero integers n, then the subgroup of G generated by a and b is a free group with $\{a, b\}$ as a freely generating set. (In our notation, $A(2) = a, B(2) = b, G = \mathrm{SL}_2(\mathbb{Z})$.)

To this end, let $X = \mathbb{Z}^2$ with $\mathrm{SL}_2(\mathbb{Z})$ acting on it the usual way, that is, by matrix multiplication with column vectors. Consider the subsets

$$C = \{(x, y) \mid |y| > |x|\}, \quad C = \{(x, y) \mid |y| < |x|\}.$$

It is clear that A and B are nonempty disjoint sets. To see $(A(2))^n(C) \subseteq D$ for $n \neq 0$, suppose that $(x, y) \in \mathbb{Z}^2$ such that $|y| < |x|$. Then $(A(2))^n(x, y) = (x, 2nx + y)$. Since $|x| > |y|$, $|2nx+y| > |2n||x|-|y| \geq 2|x|-|y| = |x|+(|x|-|y|) > |x|$. By a similar argument, we have that $(B(2))^n(D) \subseteq C$ for $n \neq 0$.

We can now apply the Ping-pong Lemma, and conclude that the subgroup of $\mathrm{SL}_2(\mathbb{Z})$ generated by $A(2)$ and $B(2)$ is free.

(2) To see that an element of $\langle A(2), B(2)\rangle$ can written in the form $(*)$, notice that the set S of invertible matrices of form $(*)$ forms a subgroup of $\mathrm{SL}_2(\mathbb{Z})$. In particular, if $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in S$, then clearly M^{-1} is also an element of S. Moreover, if $N = \left(\begin{smallmatrix} a' & b' \\ c' & d' \end{smallmatrix}\right) \in S$,

then

$$MN = \begin{pmatrix} aa' + bc' & ab' + bd' \\ a'c + c'd & b'c + dd' \end{pmatrix}.$$

Since b, c' (respectively $b'c$) are even, then $bc' \equiv 0 \pmod 4$ (respecively $b'c \equiv 0 \pmod 4$). Since a, a' (respectively d, d') are all congruent to 1 modulo 4, then $aa' \equiv dd'1 \pmod 4$. This shows that entries $aa' + bc'$ and $b'c + dd'$ are congruent to 1 modulo 4. Similarly, since b, b', c, c' are even, then $ab' + bd'$ and $a'c + c'd$ are even, so $MN \in S$, and therefore S is a subgroup of $\mathrm{SL}_2(\mathbb{Z})$.

Hence, since $A(2), A(2)^{-1}, B(2), B(2)^{-1}$ are elements of S, any element of $\langle A(2), B(2)\rangle$ can be written in the form $(*)$.

The other direction is trickier. To see an element of S is in $\langle A(2), B(2)\rangle$, we induct on the size of the largest matrix entry (in absolute value). There are many cases to consider, but in each case, multiplication by one of $A(2)$, $A(2)^{-1}$, $B(2)$, $B(2)^{-1}$ will decrease the maximum entry (in absolute value). The base case is just the identity matrix, which is obviously an element of $\langle A(2), B(2)\rangle$. So suppose all matrices with maximum entry (in absolute value) less than or equal to $k-1$ can be written as a word in $A(2)$ and $B(2)$, and consider a matrix $M\left(\begin{smallmatrix} 1+4m_1 & 2m_2 \\ 2m_3 & 1+4m_4 \end{smallmatrix}\right)$ with maximum entry (in absolute value) equal to k. Then

$$MA(2) = \begin{pmatrix} 1 + 4m_1 & 2 + 8m_1 + 2m_2 \\ 2m_2 & 1 + 4m_4 + 4m_3 \end{pmatrix},$$

$$M(A(2))^{-1} = \begin{pmatrix} 1 + 4m_1 & 2m_2 - 2 - 8m_1 \\ 2m_3 & 1 + 4m_4 - 4m_3 \end{pmatrix},$$

$$MB(2) = \begin{pmatrix} 1 + 4m_1 + 4m_2 & 2m_2 \\ 2m_3 + 2 + 8m_4 & 1 + 4m_4 \end{pmatrix},$$

$$M(B(2))^{-1} = \begin{pmatrix} 1 + 4m_1 - 4m_2 & 2m_2 \\ 2m_3 - 2 - 8m_4 & 1 + 4m_4 \end{pmatrix}.$$

Suppose additionally that $m_i > 0$, $1 \leq i \leq 4$. Then, if either the upper left or bottom left entry is the largest, multiplication on the right by $(B(2))^{-1}$ will result in a matrix with largest entry less than k. By the inductive hypothesis, that matrix can be written as a word in $A(2)$ and $B(2)$, and so M is an element of $\langle A(2), B(2) \rangle$. Similarly, if either the upper right or bottom right entry is the largest, multiplication on the right by $(A(1))^{-1}$ will reduce the largest entry to something less than k, so we conclude M is an element of $\langle A(2), B(2) \rangle$. The remaining cases for whether each m_i is positive or negative follow similarly. □

This does not say much about the *monoid* generated by these matrices, though. In fact, a generic matrix of the form above would not belong to this monoid. This is true for two reasons: first, by another result of Sanov [47], the matrices $A(2)$ and $B(2)$ generate a free group. Second, the number of matrices in the above form which are representable by positive words is negligible. In fact, the number of distinct elements represented by all freely reducible words in $A(2)$ and $B(2)$ of length $n \geq 2$ is $4 \cdot 3^{n-1}$, while the number of distinct elements represented by positive words of length $n \geq 2$ is 2^n.

Tillich and Zémor's lifting attack can still give an efficient algorithm which finds relations of length $O(\log p)$ in the group generated by $A(2)$ and $B(2)$ considered as matrices over $\mathbb{F}_p$. This algorithm is described below, but we note that it does not affect the security of the hash function based on $A(2)$ and $B(2)$ since only positive powers of $A(2)$ and $B(2)$ are used, and the group relations produced by the algorithm will involve both negative and positive powers with overwhelming probability.

Theorem 2. (Bromberg, Shpilrain and Vdovina [5, Theorem 1]) *There is an efficient heuristic algorithm that finds particular relations of the form $w(A(2), B(2)) = 1$, where w is a group word of length $O(\log p)$, and the matrices $A(2)$ and $B(2)$ are considered over $\mathbb{F}_p$.*

Proof. The following were shown in [52].

(a) For any prime p, there is an efficient heuristic algorithm that finds positive integers k_1, k_2, k_3, k_4 such that the matrix

$$\begin{pmatrix} 1 + k_1 p & k_2 p \\ k_3 p & 1 + k_4 p \end{pmatrix}$$

has determinant 1 and all k_i are of about the same magnitude $O(p^2)$.

(b) A generic matrix from part (a) has an efficient factorization (in $\mathrm{SL}(2, \mathbb{Z})$) in a product of positive powers of $A(1)$ and $B(1)$, of length $O(\log p)$. (This obviously yields a collision in $\mathrm{SL}(2, \mathbb{F}_p)$ since the matrix from part (a) equals the identity matrix in $\mathrm{SL}(2, \mathbb{F}_p)$.)

Now we combine these results with the aforementioned result of Sanov the following way. We are going to multiply a matrix from (a) (call it M) by a matrix from $\mathrm{SL}(2, \mathbb{Z})$ (call it S) with very small (between 0 and 5 by the absolute value) entries, so that the resulting matrix $M \cdot S$ has the form as in Sanov's result. Since the matrix M, by the Tillich-Zémor results, has an efficient factorization (in $\mathrm{SL}(2, \mathbb{Z})$) in a product $w(A(1), B(1))$ of positive powers of $A(1)$ and $B(1)$ of length $O(\log p)$, the same holds for the matrix $M \cdot S$. Then, since the matrix $M \cdot S$ is in "Sanov's form", we know that it is, in fact, a product of powers of $A(2)$ and $B(2)$.

Now we need one more ingredient to efficiently re-write a product of $A(1)$ and $B(1)$ into a product of $A(2)$ and $B(2)$ without blowing up the length too much. This procedure is provided by Epstein, Cannon, Holt, Levy, Paterson and Thurston in [11, Theorem 2.3.10]. We cannot explain it without introducing a lot of background material, but the fact is that, since the group $\mathrm{SL}(2, \mathbb{Z})$ is hyperbolic (whatever that means) and the subgroup generated by $A(2)$ and $B(2)$ is quasiconvex (whatever that means), there is a quadratic time algorithm (in the length of the word $w(A(1), B(1))$) that re-writes $w(A(1), B(1))$ into a $u(A(2), B(2))$

such that $w(A(1), B(1)) = u(A(2), B(2))$ and the length of u is bounded by a constant (independent of w) times the length of w.

Thus, what is now left to complete the proof is to exhibit, for all possible matrices M as in part (a) above, particular "small" matrices S such that $M \cdot S$ is in "Sanov's form". We are therefore going to consider many different cases corresponding to possible combinations of residues modulo 4 of the entries of the matrix M (recall that M has to have determinant 1), and in each case we are going to exhibit the corresponding matrix S such that $M \cdot S$ is in "Sanov's form". Denote by $\hat{M}$ the matrix of residues modulo 4 of the entries of M. Since the total number of cases is too large, we consider matrices $\hat{M}$ "up to a symmetry".

(1) $\hat{M} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

(2) $\hat{M} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$

(3) $\hat{M} = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$

(4) $\hat{M} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix}$

(5) $\hat{M} = \begin{pmatrix} 2 & 3 \\ 3 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$

(6) $\hat{M} = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$, $S = \begin{pmatrix} 0 & 1 \\ 3 & 2 \end{pmatrix}$

(7) $\hat{M} = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix}$

(8) $\hat{M} = \begin{pmatrix} 2 & 3 \\ 3 & 3 \end{pmatrix}$, $S = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$

(9) $\hat{M} = \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}$, $S = \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}$

(10) $\hat{M} = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$, $S = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$

(11) $\hat{M} = \begin{pmatrix} 3 & 0 \\ 1 & 3 \end{pmatrix}$, $S = \begin{pmatrix} -1 & 0 \\ 1 & -1 \end{pmatrix}$

(12) $\hat{M} = \begin{pmatrix} 3 & 0 \\ 2 & 3 \end{pmatrix}$, $S = \begin{pmatrix} -1 & 0 \\ 2 & -1 \end{pmatrix}$

(13) $\hat{M} = \begin{pmatrix} 3 & 0 \\ 3 & 3 \end{pmatrix}$, $S = \begin{pmatrix} -1 & 0 \\ 3 & -1 \end{pmatrix}$

(14) $\hat{M} = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix}$, $S = \begin{pmatrix} -1 & 2 \\ 2 & -5 \end{pmatrix}$

This completes the proof. □

4.3.4 Girth of the Cayley graph generated by $A(k)$ and $B(k)$

For hashing, we use only positive powers, so we need only consider products of positive powers of $A(k)$ and $B(k)$. We note that entries in a matrix of a length n product of positive powers of $A(k)$ and $B(k)$ grow fastest (as functions of n) in the alternating product of $A(k)$ and $B(k)$. This is formalized in [5, Proposition 1 and Lemma 1].

Proposition 3. *Let $w_n(a, b)$ be an arbitrary positive word of even length n, and let $W_n = w_n(A(k), B(k))$ with $k \geq 2$. Let $C_n = (A(k) \cdot B(k))^{n/2}$. Then:*

(1) *The sum of entries in any row of C_n is at least as large as the sum of entries in any row of W_n.*

(2) *The largest entry of C_n is at least as large as the sum of entries of W_n.*

Proof. First note that multiplying a matrix X by $A(k)$ on the right amounts to adding the first column multiplied by k to the second column. Similarly, multiplying on the right by $B(k)$ amounts to adding the second column multiplied by k to the first column. In particular, this means that when we build a word in $A(k)$ and $B(k)$, going from left to right, elements of the first row change independently of elements of the second row. So, we can consider only pairs of positive integers. The proposition will follow from the following lemma.

Lemma 2. *Let (x, y) be a pair of positive integers, $x \neq y$, and let $k \geq 2$. One can apply transformations of the following two kinds: transformation R takes (x, y) to $(x, y + kx)$; tranformation L takes (x, y) to $(x + ky, y)$. Among all sequences of these transformations of the same length, the sequence where R and L alternate results in:*

(1) *the largest sum of elements in the final pair;*

(2) *the largest maximum element in the final pair.*

Proof. We will prove (1) and (2) simultaneously by inducting on the length of a sequence of transformations. So suppose the lemma holds for all sequences of length at most $m \geq 2$, with the same initial pair (x, y), and suppose the final pair after m alternating transformations is (X, Y). Without loss of generality, assume that $X < Y$; this means that the last transformation applied was R. Now, applying transformation L to (X, Y) gives $(X + kY, Y)$, while applying transformation R to (X, Y) gives $(X, Y + kX)$. Since we assumed $X < Y$,

we have $X + kY > Y + kX$, so applying L results in a larger sum of elements as well as in a larger maximum element. Thus, we have a sequence of $m+1$ alternating transformations; there is one case left to consider.

Suppose some sequence of m transformations applied to (x, y) results in a pair (X', Y') with $X'+Y' < X+Y, Y' < Y$ but $X' > X$. Applying L to this pair gives $(X" + kY', Y')$ and the sum is $X'+Y'+kY' < X+Y+kY$, since $X'+Y' < X+Y$ and $Y' < Y$. The maximum element of the pair $(X'+kY', Y')$ is $X'+kY' = X'+Y'+(k-1)Y'$. Since $X'+Y' < X+Y$ and $Y' < Y$, we have $X' + kY' < X + kY$. This completes the proof of the lemma. □

The proposition follows. □

Thus we consider powers of the matrix

$$C(k) := A(k)B(k) \tag{4.3}$$

to get to entries larger than p "as quickly as possible".

4.3.5 Powers of $C(2)$

The matrix $C(2)$ is

$$\begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix}.$$

If we denote

$$(C(2))^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix},$$

then the following recurrence relations are easily proved by induction on n:

$$a_n = 5a_{n-1} + 2b_{n-1},$$
$$b_n = c_n = 2a_{n-1} + b_{n-1},$$
$$d_n = a_{n-1}.$$

Combining the recurrence relations for a_n and b_n, we get $2b_n = a_n - a_{n-1}$, so $2b_{n-1} = a_{n-1} - a_{n-2}$. Plugging this into the displayed recurrence relation for a_n gives

$$a_n = 6a_{n-1} - a_{n-2}.$$

Similarly, we get

$$b_n = 6b_{n-1} - b_{n-2}.$$

Solving these recurrence relations (with appropriate initial conditions), we get

$$a_n = \Big(\frac{1}{2} + \frac{1}{\sqrt{8}}\Big)(3+\sqrt{8})^n + \Big(\frac{1}{2} - \frac{1}{\sqrt{8}}\Big)(3-\sqrt{8})^n,$$
$$b_n = \frac{1}{\sqrt{8}}(3+\sqrt{8})^n - \frac{1}{\sqrt{8}}(3-\sqrt{8})^n.$$

Thus, a_n is the largest entry of $(C(2))^n$, and we conclude that no entry of $(C(2))^n$ is larger than p as long as $n < \log_{3+\sqrt{8}} p$. Since $C(2) = A(2)B(2)$ is a product of two generators, $(C(2))^n$ has length $2n$ as a word in the generators $A(2)$ and $B(2)$. Therefore, no two positive words of length $\leq m$ in the generators $A(2)$ and $B(2)$ (considered as matrices over $\mathbb{F}_p$) can be equal as long as

$$m < 2\log_{3+\sqrt{8}} p = \log_{\sqrt{3+\sqrt{8}}} p,$$

so we have the following.

Corollary 1. *There are no collisions of the form*

$$u(A(2), B(2)) = v(A(2), B(2))$$

if positive words u and v are of length less than $\log_{\sqrt{3+\sqrt{8}}} p$. In particular, the girth of the Cayley graph of the semigroup generated by $A(2)$ and $B(2)$ (considered as matrices over $\mathbb{F}_p$) is at least $\log_{\sqrt{3+\sqrt{8}}} p$.

The base of the logarithm here is $\sqrt{3+\sqrt{8}} \approx 2.4$. Thus, for example, if p is on the order of 2^{256}, then there are no collisions of the form $u(A(2), B(2)) = v(A(2), B(2))$ if positive words u and v are of length less than 203.

4.3.6 Powers of $C(3)$

If we consider the matrices $A(3)$ and $B(3)$ as generators of $\mathrm{SL}(2, \mathbb{F}_p)$.The matrix $C(3)$ is

$$\begin{pmatrix} 10 & 3 \\ 3 & 1 \end{pmatrix}.$$

If we let

$$(C(3))^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix},$$

and use the fact that $(C(3))^n = C(3) \cdot C(3)^{n-1}$, we get the following recurrence relations:

$$a_n = 10a_{n-1} + 3b_{n-1}$$

$$b_n = 3a_{n-1} + b_{n-1} = c_n$$

$$d_n = a_{n-1}$$

Combining these recurrence relations for a_n and b_n, we get

$$a_n = 11a_{n-1} - a_{n-2}, \quad b_n = 11b_{n-1} - b_{n-2}.$$

Solving these recurrence relations with the initial conditions $a_1 = 10$, $a_2 = 109$ and $b_1 = 3$, $b_2 = 33$, we get

$$a_n = \Big(\frac{9}{2\sqrt{117}} + \frac{1}{2}\Big)\Big(\frac{11+\sqrt{117}}{2}\Big) + \Big(\frac{1}{2} - \frac{9}{2\sqrt{117}}\Big)\Big(\frac{11-\sqrt{117}}{2}\Big),$$
$$b_n = \frac{3}{\sqrt{117}}\Big(\frac{11+\sqrt{117}}{2}\Big) - \frac{3}{\sqrt{117}}\Big(\frac{11-\sqrt{117}}{2}\Big).$$

From this we see that a_n is the largest entry of $(C(3))^n$, so no entry of $(C(3))^n$ is larger than p if $n < \log_{\frac{11+\sqrt{117}}{2}} p$. Since $C(3)$ is a product of two generators, $A(3)$ and $B(3)$, we have:

Corollary 2. *There are no collisions of the form*

$$u(A(3), B(3)) = v(A(3), B(3))$$

if positive words u and v are of length less than $2\log_{\frac{11+\sqrt{117}}{2}} p = \log_{\sqrt{\frac{11+\sqrt{117}}{2}}} p$. In particular, the girth of the Cayley graph of the semigroup generated by $A(3)$ and $B(3)$ (considered as matrices over $\mathbb{F}_p$) is at least $\log_{\sqrt{\frac{11+\sqrt{117}}{2}}} p$.

The base of the logarithm here is $\sqrt{\frac{11+\sqrt{117}}{2}} \approx 3.3$. For example, if p is on the order of 2^{256}, then there are no collisions of the form $u(A(2), B(2)) = v(A(2), B(2))$ if positive words u and v are of length less than 149.

Conclusions

We have analyzed the girth of the Cayley graph of both the group and the monoid generated by matrices $A(k)$ and $B(k)$ over $\mathbb{F}_p$ for various k. We conclude the following.

First, the lifting attack by Tillich and Zémor [52] which produces explicit relations of length $O(\log p)$ in the monoid generated by $A(1)$ and $B(1)$ can be used in conjunction with Sanov's result [47] and some results from [11] to efficiently produce relations of length $O(\log p)$ in the group generated by $A(2)$ and $B(2)$. Generically, the relations produced by this method will involve both positive and negative powers of $A(2)$ and $B(2)$. Therefore, this method does not produce collision for the corresponding hash function, since the hash function only uses positive powers of $A(2)$ and $B(2)$.

Since there is no known analog of Sanov's result for $A(3)$ and $B(3)$, at this time there is no known efficient algorithm for even producing relations of length $O(\log p)$ in the group generated by $A(3)$ and $B(3)$, let alone in the monoid. We note that by the pigeonhole principle, such relations do in fact exist.

We have computed an explicit lower bound of $\log_b p$ for the length of relations in the monoid generated by $A(2)$ and $B(2)$, where $b \approx 2.4$. For the monoid generated by $A(3)$ and $B(3)$, we have a similar lower bound with base $b \approx 3.3$.

We conclude that at this time, there are no known attacks on hash functions corresponding to the pair $A(2)$ and $B(2)$ nor on the pair $A(3)$ and $B(3)$. Therefore, there is no visible threat to their security.

Problems for future research

We list here some problems for future research on these Cayley hash functions.

Problem 1. Find a description, similar to Sanov's, for matrices in the *monoid* generated by $A(2)$ and $B(2)$ over $\mathbb{Z}$.

Problem 2. Find an analogue of "Sanov's form" for the subgroup of $\mathrm{SL}(2,\mathbb{Z})$ generated by $A(3), B(3)$.

Problem 3. Determine which words in the matrices $A(1), B(2)$ will have the fastest growth of their entries, i.e. find analogues to Proposition 3 and Lemma 2.

This problem is of interest because if we can show the alternating product again has fastest growth, then a similar calculation as was done for $A(2), B(2)$ and for $A(3), B(3)$ would show a lower bound with a smaller base. This means that the base of the logarithm is $\sqrt{2+\sqrt{3}}$, which is about 1.93. So this would mean that for p on the order of 2^{256}, there will be no collisions of the form $u(A(1), B(2)) = v(A(1), B(2))$ if positive words u and v are of length less than $269 = \log_{\sqrt{2+\sqrt{3}}}(p)$. This is a stronger bound than for either the $A(2), B(2)$ case or the $A(3), B(3)$ case.

4.4 Hashing with Burnside groups

In this section we discuss the use of matrices corresponding to homomorphisms of the Burnside group of order 3 on n generators, denoted B_n. Recall from Chapter 2 the endomorphism on B_n given by

$$a_i \mapsto a_{i+1}, \quad 1 \leq i \leq n-1,$$
$$a_n \mapsto a_1 a_n.$$

Let M_n denote the matrix corresponding (via the abelianization; see Section 2.5.1) to the above endomorphism, i.e.,

$$M_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Recall that $|M_{22}| = 15,625,959,602$ and $|M_{24}| = 10,460,353,202$. Then the order of the block matrix M, where

$$M = \begin{pmatrix} M_{22} & 0 \\ 0 & M_{24} \end{pmatrix},$$

is the least common multiple of $|M_{22}|$ and $|M_{24}|$, which is greater than 2^{65}. To hash with this matrix, let M be the hash of the 0 bit and $P^{-1}MP$ the hash of the 1 bit, where P is a random invertible 46 by 46 matrix. Recall that since the abelianization is a homomorphism, the order of the matrix here divides the order of the corresponding endomorphism. In particular, the order of the endomorphism has 2^{65} as a divisor.

4.5 Computations and efficiency

In this section we include results of some experiments done to test the efficiency of the proposed hashes. On the one hand, we consider hashing with a 46×46 matrix over $\mathbb{Z}_3$ (see Section 4.4) and on the other hand we hash with 2×2 matrices over $\mathbb{F}_p$ for a large prime p.

We conducted several tests, performed on a computer with an Intel Core i7 quad-core 4.0 GHz processor and 16 GB of RAM, running Linux Mint version 17.1 with Python version 3.4.1, NumPy version 1.9.1.

Since we are working within the abelianization of the Burnside groups B_n of exponent 3, the entries of the matrices are essentially elements of $\mathbb{Z}_3$. This means we are not really concerned with reducing modulo a prime p, since p is so small, but rather the multiplication is done with use of a simple multiplication table. If the exponent were larger, we would need to take reduction modulo the exponent into account for computation.

Using GAP [14], we generated a random 46×46 matrix M over $\mathbb{Z}_3$, and noted how long it took to compute $M^{10,000}$. The result was that it took, on average, 1000 milliseconds. Since GAP is not the most efficient computer system for matrix multiplication (it is a far better tool for more complicated group algorithms), compare this to the time for the same computation using Python: 970 milliseconds.

Working with 2×2 matrices over a large field $\mathbb{F}_p$ for large prime p, we note that multiplication of the matrices themselves is quite fast (can be done in 7 multiplications), but reduction modulo p takes more work. To test the efficiency with multiplication in $\mathrm{SL}_2(\mathbb{F}_p)$, we conducted two experiments, both with $p = 2^{127} - 1$. In the first, we chose a random number between 1 and $1,000,000$, found a matrix M as a word in $A(2)$ and $B(2)$ of that length, then computed that it took approximately 80 milliseconds to compute $M^{10,000}$. In the second experiment, we determined that it took approximately 30 milliseconds to compute a matrix as a word of length $10,000$ in $A(2)$ and $B(2)$ over $\mathbb{F}_{2^{127}-1}$.

For comparison, see [8] for performance results of various cryptographic functions. In particular, SHA-512 hashes approximately 99 MiB/second (MiB stands for mebibyte, and 1 MiB $= 2^{20}$ bytes) and so this is roughly 10^8 bytes per second. Our proposed hash (the second experiment) also hashes approximately 10^8 bytes per second. Moreover, SHA-512 has been optimized; our hash performs at this speed without any optimization. For instance, our computations involve performing the reduction modulo p at each step.

Also, our computation can be parallelized, whereas SHA-512 (and others in the SHA family) cannot. This is because our bitstrings can be broken up into smaller parts, hashed,

and then "put back together:" for instance, if H denotes the hash function, and the message $M = ABC$, then $H(M) = H(ABC) = H(A)H(B)H(C)$. This is not true with SHA hashes.

We conclude that using small matrices over large field yields faster multiplication time than using large matrices over a small field. It is hard to say, however, what particular algorithms NumPy and GAP use to reduce modulo prime numbers, i.e. whether (and how) these programs capitalize on the efficiency of "reduction modulo 3" and multiplication of 2×2 matrices.

Bibliography

[1] I. Anshel, M. Anshel, and D. Goldfeld. An algebraic method for public-key cryptography. *Math. Res. Lett.*, 6(3-4):287–291, 1999.

[2] I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux. Key agreement, the Algebraic Eraser$^{\mathrm{TM}}$, and lightweight cryptography. In *Algebraic methods in cryptography*, volume 418 of *Contemp. Math.*, pages 1–34. Amer. Math. Soc., Providence, RI, 2006.

[3] G. Baumslag, N. Fazio, A. R. Nicolosi, V. Shpilrain, and W. E. Skeith, III. Generalized learning problems and applications to non-commutative cryptography (extended abstract). In *Provable security*, volume 6980 of *Lecture Notes in Comput. Sci.*, pages 324–339. Springer, Heidelberg, 2011.

[4] J. Bourgain and A. Gamburd. Uniform expansion bounds for Cayley graphs of $\mathrm{SL}_2(\mathbb{F}_p)$. *Ann. of Math. (2)*, 167(2):625–642, 2008.

[5] L. Bromberg, V. Shpilrain, and A. Vdovina. Navigating the cayley graph of $\mathrm{SL}_2(\mathbb{F}_p)$ and applications to hashing. submitted.

[6] P. Camion. Can a fast signature scheme without secret key be secure? In *Applied algebra, algorithmics and error-correcting codes (Toulouse, 1984)*, volume 228 of *Lecture Notes in Comput. Sci.*, pages 215–241. Springer, Berlin, 1986.

[7] R. Coulon. A criterion for detecting trivial elements of Burnside groups.

[8] W. Dai. Crypto++ 5.6.0 benchmarks.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

[10] J. Ding, A. Miasnikov, and A. Ushakov. A linear attack on a key exchange protocol using extensions of matrix semigroups.

[11] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

[12] B. Fine, M. Habeeb, D. Kahrobaei, and G. Rosenberger. Aspects of nonabelian group based cryptography: a survey and open problems. *JP J. Algebra Number Theory Appl.*, 21(1):1–40, 2011.

[13] R. W. Floyd. Assigning meanings to programs. In *Proc. Sympos. Appl. Math., Vol. XIX*, pages 19–32. Amer. Math. Soc., Providence, R.I., 1967.

[14] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.7.2*, 2013.

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

[16] M. Habeeb, D. Kahrobaei, C. Koupparis, and V. Shpilrain. Public key exchange using semidirect product of (semi)groups. *Lecture Notes Comp. Sc.*, 7954:475–486, 2013.

[17] M. Hall, Jr. *The theory of groups*. The Macmillan Co., New York, N.Y., 1959.

[18] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Professional Computing. Springer-Verlag, 2004.

[19] G. Havas, M. F. Newman, A. C. Niemeyer, and C. C. Sims. Groups with exponent six. *Comm. Algebra*, 27(8):3619–3638, 1999.

[20] H. A. Helfgott. Growth and generation in $\mathrm{SL}_2(\mathbb{Z}/p\mathbb{Z})$. *Ann. of Math. (2)*, 167(2):601–623, 2008.

[21] J. Hughes and A. Tannenbaum. Lenght-based attacks for certain group based encryption rewriting systems, 2002. Workshop SECI02 Securite de la Communication sur Internet.

[22] S. V. Ivanov. The free Burnside groups of sufficiently large exponents. *Internat. J. Algebra Comput.*, 4(1-2):ii+308, 1994.

[23] D. Kahrobaei and M. Anshel. Decision and search in non-abelian Cramer-Shoup public key cryptosystem. *Groups Complex. Cryptol.*, 1(2):217–225, 2009.

[24] D. Kahrobaei and B. Khan. A noncommutative generalization of el-gamal key exchange using polycyclic groups. Number 4 in Proceedings of the Global Telecommunications Conference. 2006.

[25] D. Kahrobaei and C. Koupparis. Non-commutative digital signatures. *Groups Complex. Cryptol.*, 4(2):377–384, 2012.

[26] D. Kahrobaei, C. Koupparis, and V. Shpilrain. Public key exchange using matrices over group rings. *Groups Complex. Cryptol.*, 5(1):97–115, 2013.

[27] D. Kahrobaei, H. Lam, and V. Shpilrain. Public key exchange using extensions by endomorphisms and matrices over a galois field. Preprint.

[28] D. E. Knuth. *The art of computer programming. Vol. 2: Seminumerical algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1969.

[29] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J.-s. Kang, and C. Park. New public-key cryptosystem using braid groups. In *Advances in cryptology—CRYPTO 2000 (Santa Barbara, CA)*, volume 1880 of *Lecture Notes in Comput. Sci.*, pages 166–183. Springer, Berlin, 2000.

[30] M. Kreuzer, A. D. Myasnikov, and A. Ushakov. A linear algebra attack to group-ring-based key exchange protocols. In *Applied cryptography and network security*, volume 8479 of *Lecture Notes in Comput. Sci.*, pages 37–43. Springer, Cham, 2014.

[31] J. Lafferty and D. Rockmore. Numerical investigation of the spectrum for certain families of Cayley graphs. In *Expanding graphs (Princeton, NJ, 1992)*, volume 10 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 63–73. Amer. Math. Soc., Providence, RI, 1993.

[32] H. Lam. *Exploring platform (semi)groups for noncommutative key-exchange protocols*. PhD thesis, Graduate Center of the City University of New York, 2014.

[33] A. Lubotzky. *Discrete groups, expanding graphs and invariant measures*, volume 125 of *Progress in Mathematics*. Birkhäuser Verlag, Basel, 1994. With an appendix by Jonathan D. Rogawski.

[34] G. A. Margulis. Explicit constructions of graphs without short cycles and low density codes. *Combinatorica*, 2(1):71–78, 1982.

[35] G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredachi Informatsii*, 24(1):51–60, 1988.

[36] A. Miasnikov and V. Roman'kov. A linear decomposition attack.

[37] A. Myasnikov, V. Roman'kov, A. Ushakov, and A. Vershik. The word and geodesic problems in free solvable groups. *Trans. Amer. Math. Soc.*, 362(9):4655–4682, 2010.

[38] A. Myasnikov, V. Shpilrain, and A. Ushakov. *Group-based cryptography*. Advanced Courses in Mathematics. CRM Barcelona. Birkhäuser Verlag, Basel, 2008.

[39] A. Myasnikov, V. Shpilrain, and A. Ushakov. *Non-commutative cryptography and complexity of group-theoretic problems*, volume 177 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2011. With an appendix by Natalia Mosina.

[40] C. Petit. *Cryptographic hash functions from expander graphs*. Phd thesis, Uuniversity College London, 2009.

[41] C. Petit, K. Lauter, and J.-J. Quisquater. Cayley hashes: A class of efficient graph-based hash functions. preprint.

[42] C. Petit and J.-J. Quisquater. Preimages for the Tillich-Zémor hash function. In *Selected areas in cryptography*, volume 6544 of *Lecture Notes in Comput. Sci.*, pages 282–301. Springer, 2011.

[43] C. Petit and J.-J. Quisquater. Preimages for the Tillich-Zémor hash function. In *Selected areas in cryptography*, volume 6544 of *Lecture Notes in Comput. Sci.*, pages 282–301. Springer, Heidelberg, 2011.

[44] C. Petit and J.-J. Quisquater. Rubik's for cryptographers. *Notices Amer. Math. Soc.*, 60(6):733–740, 2013.

[45] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Math. Comp.*, 32(143):918–924, 1978.

[46] V. Roman'kov. Linear decomposition attack on public key exchange protocols using semidirect products of (semi)groups.

[47] I. N. Sanov. A property of a representation of a free group. *Doklady Akad. Nauk SSSR (N. S.)*, 57:657–659, 1947.

[48] P. Sarnak. *Some applications of modular forms*, volume 99 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 1990.

[49] V. Shpilrain and A. Ushakov. The conjugacy search problem in public key cryptography: unnecessary and insufficient. *Appl. Algebra Engrg. Comm. Comput.*, 17(3-4):285–289, 2006.

[50] V. Shpilrain and G. Zapata. Using the subgroup membership search problem in public key cryptography. In *Algebraic methods in cryptography*, volume 418 of *Contemp. Math.*, pages 169–178. Amer. Math. Soc., Providence, RI, 2006.

[51] E. Teske. A space efficient algorithm for group structure computation. *Math. Comp.*, 67(224):1637–1663, 1998.

[52] J.-P. Tillich and G. Zémor. Group-theoretic hash functions. In *Algebraic coding (Paris, 1993)*, volume 781 of *Lecture Notes in Comput. Sci.*, pages 90–110. Springer, Berlin, 1994.

[53] J.-P. Tillich and G. Zémor. Hashing with SL_2. In *Advances in cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Comput. Sci.*, pages 40–49. Springer, Berlin, 1994.

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