Cryptographic Hash Functions and Some Applications to Information Security

Lisa Bromberg

Abstract We explore hashing with matrices over $\mathrm{SL}_2(\mathbb{F}_p)$, outlining known results of Tillich and Zémor. We then summarize the bounds on the girth of the Cayley graph of the subgroup of $\mathrm{SL}_2(\mathbb{F}_p)$ for specific generators A, B, work done by the author, Shpilrain, and Vdovina. We demonstrate that even without optimization, these hashes have comparable performance to hashes in the SHA family.

Keywords Information security · Cryptography · Group theory

1 Introduction

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 (asymmetric)* 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 [4]. 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

L. Bromberg (✉)
United States Military Academy, West Point, NY, USA
e-mail: lisa.bromberg@usma.edu

M. B. Nathanson (ed.), *Combinatorial and Additive Number Theory II*,
Springer Proceedings in Mathematics & Statistics 220,
https://doi.org/10.1007/978-3-319-68032-3_6

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 asymmetric cipher to produce a session key and then proceed with symmetric encryption for the remainder of the session.

1.1 Hash Functions

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 bit string, 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.

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 outline results of the authors, Shpilrain and Vdovina [1] using matrices over $\mathrm{SL}(2, \mathbb{F}_p)$ of a particular form which generate the group.

Definition 1 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 [11].

Another direction, more relevant to our exposition, 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 [16]. However, this newer hash function is also vulnerable to attack; see [17]. 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 Sect. 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 1), 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.

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 bit string 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 [16] in the case where two matrices A and B generate (as a monoid) all of $\mathrm{SL}(2, \mathbb{Z}_+)$. 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; see [1]. Since only positive powers are used for hashing, this is all that is needed, and so, 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 recall lower bounds on the length of collisions for hash functions corresponding to some particular pairs of matrices from $\mathrm{SL}(2, \mathbb{F}_p)$; again, see [1].

1.2 Cayley Hash Functions

Classical hash functions mix pieces of the message repeatedly so the result appears sufficiently random [13]. 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 with 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 2 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 [16].

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 [17, 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 [18].

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

Definition 3 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 [8]. 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. 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 10–20 times slower than the standard classical hash SHA. Computation in Cayley hashes can be easily parallelized, which could be a major benefit in applications. We outline a hash function based on the Tillich–Zémor hash function [1] which is resistant to known methods of attack and which is efficient in computation.

1.3 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 [2], 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. For a specific example of how the lifting attack works in the case of the Tillich–Zémor hash function, see [16].

2 Hashing with Matrices

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 1 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* [17].

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 yield 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 [15], Lafferty and Rockmore [7], and Margulis [9, 10].

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. The bound is on the order of $\log p$.

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 $B(n)$.

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 [16]. 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 [16] describe a probabilistic algorithm 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 [17] for details on the implementation 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

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

This hash function was published in 1994 [17], but there have been several recent attacks. In 2010, Petit and Quisquater [12] describe a preimage attack; in 2011, Grassl, Ilic, Magliveras and Steinwandt [6] describe a collision attack.

3.2 *Hashing with* $A(2)$ *and* $B(2)$

In this section, we outline circuits in the Cayley graph of $\mathrm{SL}(2, \mathbb{F}_p)$ with generating set $A(2)$, $B(2)$, as presented in [1]. 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. 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 $B(2)$.

Theorem 1 Sanov [14]

(1) The group generated by

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

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*

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

where the m_i *are integers.*

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 [14], 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 $\mathbb{F}_p$. 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 [1, 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* $\mathbb{F}_p$.

3.3 Girth of the Cayley Graph Generated by $A(k)$ and $B(k)$

For hashing, we use only positive powers, so we need only to 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 faster (as functions of n) in the alternating product of $A(k)$ and $B(k)$. This is formalized below.

Proposition 2 ([1, Proposition 1]) *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.

Lemma 1 ([1, Lemma 1]) *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* $(x, y + kx)$; *transformation 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.

Thus, we consider powers of the matrix

$$C(k) := A(k)B(k) \tag{1}$$

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

3.3.1 Powers of $C(2)$

As seen in the work of the authors, Shpilrain and Vdovina [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.

3.3.2 Powers of $C(3)$

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

3.4 Conclusions

First, the lifting attack by Tillich and Zémor [16] 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 [14] and some results from [5] 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.

3.5 Problems for Future Research

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

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

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

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

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 Computations and Efficiency

In this section, we include results of some experiments done to test the efficiency of the hashes proposed in [1]. 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 and NumPy version 1.9.1.

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, and then computed that it took approximately 80 ms 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 [3] 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 bit strings 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.

References

1. L. Bromberg, V. Shpilrain, A. Vdovina, Navigating the cayley graph of $\mathrm{SL}_2(\mathbb{F}_p)$ and applications to hashing. Semigroup Forum **94**, 314–324 (2017)
2. 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 Computer Science (Springer, Berlin, 1986), pp. 215–241
3. W. Dai, Crypto++ 5.6.0 benchmarks
4. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976
5. D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson, W.P. Thurston, *Word Processing in Groups* (Jones and Bartlett Publishers, Boston, MA, 1992)
6. M. Grassl, I. Ilić, S. Magliveras, R. Steinwandt, Cryptanalysis of the Tillich-Zémor hash function. J. Cryptol. **24**(1), 148–156 (2011)
7. J. Lafferty, D. Rockmore, Numerical investigation of the spectrum for certain families of Cayley graphs, in *Expanding Graphs (Princeton, NJ, 1992)*, volume 10 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science (American Mathematical Society, Providence, RI, 1993), pp. 63–73
8. 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
9. G.A. Margulis, Explicit constructions of graphs without short cycles and low density codes. Combinatorica **2**(1), 71–78 (1982)
10. 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)
11. C. Petit, Cryptographic hash functions from expander graphs. Ph.D. thesis (University College London, 2009)
12. C. Petit, J.-J. Quisquater, Preimages for the Tillich-Zémor hash function, in *Selected Areas in Cryptography*, volume 6544 of Lecture Notes in Computer Science (Springer, 2011), pp. 282–301
13. C. Petit, J.-J. Quisquater, Rubik's for cryptographers. Not. Am. Math. Soc. **60**(6), 733–740 (2013)
14. I.N. Sanov, A property of a representation of a free group. Doklady Akad. Nauk SSSR (N. S.) **57**, 657–659 (1947)
15. P. Sarnak, *Some Applications of Modular Forms*. Cambridge Tracts in Mathematics, vol. 99 (Cambridge University Press, Cambridge, 1990)

16. J.-P. Tillich, G. Zémor, Group-theoretic hash functions, in *Algebraic Coding (Paris, 1993)*, volume 781 of Lecture Notes in Computer Science (Springer, Berlin, 1994), pp. 90–110
17. J.-P. Tillich, G. Zémor, Hashing with SL_2, in *Advances in Cryptology—CRYPTO '94*, volume 839 of Lecture Notes in Computer Science (Springer, Berlin, 1994), pp. 40–49
18. G. Zémor, Hash functions and Cayley graphs. Des. Codes Cryptogr. **4**(4), 381–394 (1994)