SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

# An Introduction to Cayley Hash Functions

av

**Tove Gertonsson**

2021 - No K14

# An Introduction to Cayley Hash Functions

Tove Gertonsson

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Rikard Bögvad

2021

## 1 Introduction

This paper will explore Cayley hash functions, which - as the name suggests - are a type of hash function based on Cayley graphs. These hash functions are, unlike for example the SHA family of hash functions, not block ciphers but each bit is hashed individually, and the hash value itself corresponds to a walk on a Cayley graph. We will in particular be looking at Zémor's first proposal of such a function from 1991, which uses matrices of $SL(2, \mathbb{F}_p)$ for hashing. Further we will see how this first version was broken by Zémor and Tillich, whose lifting attack is based on the rather simple Euclid's algorithm. We will also have a look at Bromberg's suggestions on improvement of Zémor's first proposal. To begin with, however, we will briefly introduce the concept of hash functions in general and cryptographic hash functions in particular, and in what way the two differ in terms of security requirements. Further, as Cayley hash functions actually rest on mathematical theory and structure, we shall proceed by getting familiar with the group theory needed to understand the mechanics at work in these hash functions. We will also add the graph-theoretic notions "girth" and "expander graph" to our vocabulary, in order to provide a little more thorough understanding of the security components at hand when hashing with Cayley graphs.

## 2 Hash Functions

### 2.1 Definition

Whenever we are storing plenty of data and wish to access a certain part of it, time consumption is one of the greatest obstacles to tackle. While we would prefer the fetching of data to run fast and smoothly, searching through an unsorted (or even a sorted) array by looping through its indexes is a process that quickly explodes in time complexity. One way around such a problem is to associate each data with a *key*, which is transformed by some function into a more handy size. The transformed key can then be used as an index or address in a table, in which we store the data connected to that particular key. To fetch a certain data, we only need to apply our function to the key corresponding to the data (which gives us the index where the data is stored), and we can go directly to the correct index. The complexity for accessing a value with known index is only $O(1)$ - in other words a constant amount of time.

The type of table described above is a *hash table*, and the function used to produce the index by which we sort our data is a type of *hash function*. The hash function in the above example would transform an input of arbitrary size (number of bits), i.e. the key, into an output of fixed size, i.e. the index or hash, and then store the keys and the values in the hash table. In fact, put in more formal and general terms this is exactly what hash functions are; they are functions that map data of arbitrary length to an output of fixed length. The output data is usually referred to as hash code, digest hashes or hash values. The function input is taken as a key, which is used to identify some specific information or data, or it can consist of the data itself.[1] Further, the function is designed in such a way that two equal keys must return the same hash value. There is, however, no guarantee that different keys will result in different hash values, which means that collisions are, at least in theory, possible. There are three main purposes for a hash function:

1. To convert arbitrary length keys into hash values of fixed length.

2. To produce uniformly distributed hash values over the key space. (For each key, any hash value is equally likely.)

3. To produce a value whose length is shorter or equal in length to the input keys.

A well-designed hash function should also satisfy the following properties:

4. It is fast and easy to compute, roughly linear time.

5. It minimizes the amount of collisions.

In order to provide a more intuitive understanding of how hash functions operate, I will walk you through an everyday example of a hash function, the International Standard Book Number.

**Example 2.1** (The International Standard Book Number)**.** The ISBN is in fact a type of hash value, intended to be unique for each book and separate edition, meaning that an e-book, hardcover and paperback edition of the same book will have different hash values. That way, the ISBN works as a product identifier and is used by for example publishers and libraries for ordering, listing and stock control. The 13-digit ISBN is made up of the following structure[2]:

1. A pre-fix element; this far 987 or 979 is used.

2. The registration group element, identifying the particular language area, individual country or territory. It is constituted of 1 to 5 digits.

3. The registrant element, identifying the particular publisher or imprint.

[1]Consider for example a set of passwords that are stored as their corresponding hash values; we would rather have somebody getting hold of these than the actual passwords.

[2]Note that although the lengths of the different elements may vary, the total number of digits is always 13.

Constituted of maximum 7 digits.

4. The publication element, which identifies the particular edition and format of a specific title. Up to 6 digits long.

5. A checksum character or check digit. This is a final single digit that validates the whole hash code.

We might look upon the *entire* ISBN as a hash value, but the last check digit $r$ actually also constitutes a hash value by itself. This last check digit ranges from 0 to 9 and is chosen so that the sum of all 13 digits each multiplied by its integer weight, alternating between 1 and 3, is a multiple of 10. Deciding the value of the 13-th digit is done by calculating the following hash function.

Let

$$r = 10 - (x_1 + 3x_2 + x_3 + 3x_4 + ... + x_{11} + 3x_{12}) \pmod{10},$$

then we get the following 13-th hash digit

$$x_{13} = \begin{cases} r; & \text{if } r < 10, \\ 0; & \text{if } r = 10. \end{cases}$$

Obviously, there is no need for this system to be secretive by hiding or making it hard to obtain the above listed information about the book which produces a certain last hash value. We will see that the need for secrecy is the main difference between general hash functions and *cryptographic* hash functions.

## 3 Cryptographic Hash Functions

Like the general hash function, the cryptographic hash function is a one-way function which maps data of arbitrary size into a bit array of fixed size. "One-way" translates to mathematical terms as "practically infeasible to invert". It differs from the normal two-way *encryption* functions, as it does not preserve entire plaintexts. Plaintext means in this context the input of a hash function, i.e. a message or text that has not yet been encrypted. Further, there is no ensurance of the function being theoretically invertible, i.e. injective, meaning that it is in reality possible for a collision to occur, just like with normal hash functions. As opposed to normal hash functions however, a cryptographic hash function is required to fulfill a number of properties regarding security.

Cryptographic hash functions are used frequently in user authentication such as for digital signatures and passwords, making them a predominant part of encryption mathematics.

While the purposes of a general hash function are true also for cryptographic hash functions, cryptographic hash functions have, as mentioned, a more rigid set of requirements concerning security. Below we denote by $\{0,1\}^k$ the set of binary words of length $k$ (i.e. constituted by $0's$ and $1's$) and by $\{0,1\}^*$ binary words of arbitrary length. While the first requirement assures that inverting the function is difficult, the last two requirements deal with the possible dilemmas that may arise from the fact that hash functions are not injective. Since collisions can occur, there is a need to prevent more or less trivial such from occurring.

**Requirements for cryptographic hash functions** *Let $n \in \mathbb{N}$ and let $H : \{0,1\}^* \to \{0,1\}^n$. The hash function $H$ is required to satisfy the following:*

1. ***Preimage resistance*** *Given output $y$, it is computationally infeasible*[3] *to find input $x$ such that $H(x) = y$.*

2. ***Second preimage resistance*** *Given input $x$, it is computationally infeasible to find $y \neq x$ such that $H(x) = H(y)$. This property is sometimes referred to as weak collision resistance.*

3. ***Collision resistance*** *It is computationally infeasible to find inputs $x \neq y$ such that $H(x) = H(y)$. This property is sometimes referred to as strong collision resistance.*

The need for pre-image resistance is quite easy to understand intuitively in the sense that without pre-image resistance, an adversary could for example figure out how to produce the unique digital signature of somebody else. They can then use the signature to sign fake documents with, making the receiver of the document believe it is from the real author of the signature.

The lack of injectivity might cause a scenario where two different inputs result in the same hash value. Therefore, the second pre-image resistance and collision resistance properties are needed to avoid more trivial non-injective functions. The following example from [7] clearly illuminates what type of situations a lack in these resistances can cause. Suppose we have an adversary who is able to find two documents, $D_1$ and $D_2$, such that $H(D_1) = H(D_2)$. Suppose $D_1$ says "Pay the bearer 10 dollars" and $D_2$ says "Pay the bearer 100 dollars". The adversary can then give an unsuspecting person 10 dollars and ask her to sign $D_1$. But, as the unsuspecting person has actually signed the hash value $H(D_1)$, she has

[3]If a problem is *computationally infeasible* it means that it can not be solved with all existing computing power in the world.

simultaneously signed $H(D_2)$. Our evil adversary can then go to the bank, show the banker the document $D_2$ together with its signed hash, and then get paid 100 dollars [7].

The logic behind referring to collision resistance as strong collision resistance and to second preimage as weak collision resistance, can be understood with the birthday paradox. Finding a collision with a *specific* birthday amongst $n$ random people is much less likely than finding *any two* colliding birthdays amongst the $n$ people. Hence, a function fulfilling the collision resistance property automatically fulfills the second preimage resistance; if it is difficult to find *any two* colliding birthdays, the task of finding a collision with a specific one is even more daunting. (However, fulfilling the strong collision resistance requirement does not imply that the preimage resistance is fulfilled.)

Adding to the requirements above, an ideal cryptographic hash function should also have the following properties.

1. It should be quick to compute the hash value for any given message, roughly linear time.

2. Any small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value (avalanche effect).

Our initial example with the last check digit of the ISBN number, is a good example of a hash function that is *not* a cryptographic hash function as the purpose of the last check digit is merely to verify the ISBN number. Producing a collision is uncomplicated; given an ISBN, we can create a new ISBN with the first 12 digits corresponding to the same congruence classes modulo 10 as our initial ISBN. This way we will have created a collision and an ISBN that would evaluate to "true" - regardless of wether our book exists or not.

## 4 Free Monoids

Before exploring the main subject of this paper, we will introduce some algebraic terms necessary for understanding the structures of Cayley hash functions; free monoids, free groups and Cayley graphs. We will first look at the algebraic structure of free monoids, which have the following definition.

**Definition 4.1.** *A monoid is a set $S$ together with a binary operation $S \times S \to S$ such that for $a, b \in S$ $(a, b) \mapsto a \cdot b$, satisfying the following:*

1. ***Associativity*** *For all elements $a$, $b$ and $c$ in $S$, it holds that $(a \cdot b) \cdot c =$*

$a \cdot (b \cdot c)$.

2. ***Identity element*** *There exists an element $e$ in $S$ such that $a \cdot e = e \cdot a = a$ for all elements $a$ in $S$.*

A *free monoid* on a set is defined as the monoid over elements consisting of all possible finite sequences of zero or more elements from the set. The binary operation of the free monoid is *string concatenation*, which means that strings are joined end-to-end. The identity element is set to the empty string; adding an empty string to any word will not change the original string. More formally, a monoid is free if it is isomorphic to the free monoid on a set. As an example, we may consider the free monoid on two generators, where we have that $S = \{a, b\}$ for some $a, b$ is the generating set of the monoid. The elements of this monoid consist of all possible words or combinations that can be formed using $S$ as an alphabet, resulting in words such as *aaba* or *bba*. If we let $a = 1$ and $b = 0$, we have obtained the monoid on binary numbers consisting of words such as 0010, 101, 111 etc.

Consider now the free monoid, $F$, of the single generator 1, and concatenation represented by a plus sign. This monoid consists of the sequences "1", "1+1", "1+1+1" etc, with the empty word as the identity. We can then map each of these sequences to their evaluation result and the empty sequence to 0 in $M = (\mathbb{N}_0, +)$, which results in the following homomorphism:

$$\varphi\colon\ F \to M.$$

We have that if $s$ is mapped to the natural number $m$ and $t$ is mapped to the natural number $n$, $s + t \mapsto m + n$ (i.e. the sum of the natural numbers $m$ and $n$). This is clearly a homomorphism, since taking the string $s + t \in F$ we get:

$$\varphi(s + t) = m + n = \varphi(s) + \varphi(t).$$

This map is injective, since for any specific sequence of ones there will be one exact corresponding natural number. It is also clearly surjective, since any number in $\mathbb{N}_0$ can be represented as a unique number of ones. Hence, $M = (\mathbb{N}_0, +)$ is a free monoid.

## 5 Free Groups

We are now ready to expand our group-theoretic knowledge to *free groups*. Let $S = \{s_1, \dots\}$ be a set and $S^{-1} = \{s^{-1}, \dots\}$ a set that is disjoint to $S$, and simultaneously in 1-to-1 correspondence with $S$. Further, let $\tilde{F_S}$ be the set of all possible words that can be formed using $S \cup S^{-1}$ as an alphabet. Then $\tilde{F_S}$ is a monoid under the binary operation concatenation, with the empty word as identity. Now, let us consider the smallest equivalence relation on $\tilde{F_S}$ such that

$$xs_is_i^{-1}y \sim xy,$$

$$xs_i^{-1}s_iy \sim xy$$

for all $x, y, s_i \in S$. This implies that if $x = y$ equals the empty string we have that

$$ss^{-1} \sim e,$$

$$s^{-1}s \sim e.$$

We clearly have that $x \sim x'$ and $y \sim y'$ implies that $xy \sim x'y'$, showing that the choice of representatives does not matter. Further, the set of equivalence classes, $F_S$, forms a group since each element $s_k^{\pm}...s_i^{\pm}s_j^{\pm}$ has an inverse in $s_j^{\mp}s_i^{\mp}...s_k^{\mp}$, using the equivalence relation $\mp = -\pm$. Each equivalence class contains one *reduced* word $x = s_1s_2\dots s_k$, such that $s_{i+1} \neq s_i^{-1}$ for all $i < k$. By definition, we have that two reduced words $s_1^{\epsilon_1}s_2^{\epsilon_2}\dots s_n^{\epsilon_n}$ and $r_1^{\delta_1}r_2^{\delta_2}\dots r_n^{\delta_m}$ , where $\epsilon_i \pm 1$ and $\delta_i = \pm 1$ for all $i$, are equal if and only if $n = m$ and $\delta_i = \epsilon_i$, $1 \leq i < n$. From now on we will only consider the reduced word from each equivalence class. When concatenating two of these words, we will continue to only consider reduced words. This means substituting all occurrences of $ss^{-1}$ or $s^{-1}s$ with the empty string, until we have reached a reduced word on the above form. This group of equivalence classes under concatenation, $F_S$, is called the *free group* on the set $S$. More generally, a group $G$ is called free if it can be generated by some subset $S$ of $G$, and is isomorphic the the free group on this set, $F_S$. The notion of a group $G$ being free and generated by $S$ can be formalized by the following universal property.

**Theorem 5.1.** *Let $G$ be a group with generating set $S \subset G$. Then $G$ is free on $S$ if and only if the following holds: every map $\varphi : S \to H$ from $S$ into some group $H$ can be extended to a unique homomorphism $f : G \to H$, making the below diagram commute:*

$$\begin{tikzcd} & G \arrow[dd, dashed, "f"] \\ S \arrow[ur, "i"] \arrow[dr, "\varphi"'] & \\ & H \end{tikzcd}$$

*(where $i : S \to G$ is the inclusion of $S$ into $G$).*

One example of a free group is the non-abelian group on two generators under concatenation, i.e. with generating set $S = \{a, b\}$ and inverse set $S^{-1} = \{a^{-1}, b^{-1}\}$. Since the group operation in $F_S$ is concatenation, the elements of the group are all possible combinations from the generating set, such as $aaab^{-1}abb$ etc. If we add a single relation to the this group, by:

$$ab = ba$$

we have created a free *abelian* group on the same generating set, and thereby we are allowed to change the order of the elements in a word. This results in words on the form $a^k b^{k'}$ where $k, k' \in \mathbb{Z}$, as in fact *all* elements commute in this group. We can see this with only a few operations; multiplying $ab = ba$ on the left with $a^{-1}$ results in $b = a^{-1}ba$, and by multiplying on the right with $a^{-1}$ once more we obtain $ba^{-1} = a^{-1}b$. In a similar manner, we can show commutativity between any two elements, and we can draw the conclusion that all elements in the free abelian group on two generators are on the above form. For example, the above expression is by the rule of commutativity identified as $aaab^{-1}abb = a^4b$.

We will now have a look at an example of a free abelian group, $\mathbb{Z} \times \mathbb{Z}$ under addition with generating set $X = \{(1, 0), (0, 1)\}$. We will show that this group is free by using theorem 5.1, and finding an isomorphism between $\mathbb{Z} \times \mathbb{Z}$ and the free abelian group on two generators, $A_S$. Hence, let $F_S$ be the free (non-abelian) group with generating set $S = \{a, b\}$. Let

$$i : S \to F_S$$

be the inclusion of $S$ into $F_S$. Consider the map:

$$\varphi : S \to (\mathbb{Z} \times \mathbb{Z}, +), \text{ s.t. } a \mapsto (1,0) \text{ and } b \mapsto (0,1).$$

From theorem 5.1, we can now extend the map $\varphi$ to a unique homomorphism from $F_S$ to $\mathbb{Z} \times \mathbb{Z}$:

$$f : F_S \to (\mathbb{Z} \times \mathbb{Z}, +), \text{ s.t. } w \mapsto (k - j, k' - j'),$$

where $k$ equals the number of $a$:s occurring in $w$, $k'$ the number of $b$:s, $j$ the number of $a^{-1}$:s and $j'$ the number of $b^{-1}$:s [4]. Using the graph from theorem 5.1 to illustrate the above, we have now arrived at the following map:

$$\begin{tikzcd} & F_S \arrow[dd, dashed, "f"] \\ S \arrow[ur, "i"] \arrow[dr, "\varphi"'] & \\ & (\mathbb{Z} \times \mathbb{Z}, +) \end{tikzcd}$$

Further, we have that $f$ factors through the free abelian group on two generators, $A_S$. This means that there are maps $\theta$ and $\psi$ such that

$$f = \theta \circ \psi$$
$$\text{where } \theta : F_S \to A_S \text{ and } \psi : A_S \to (\mathbb{Z} \times \mathbb{Z}).$$

As we saw previously, due to the commutativity of $A_S$, elements in this group are on the form $a^k b^{k'}$. Hence, $\psi$ maps an element $a^k b^{k'}$ to $(k, k')$. From this fact we can easily see that $\psi$ is surjective; given any $(k, k') \in (\mathbb{Z} \times \mathbb{Z})$, there is an element $a^k b^{k'}$ that maps to it. It is also clearly injective, since if we have that $\psi(a^k b^{k'}) = \psi(a^j b^{j'})$ we must have $(k, k') = (j, j')$, implying that $k = j$ and $k' = j'$. Hence, $(\mathbb{Z} \times \mathbb{Z}, +)$ is isomorphic to $A_S$ and therefore a free abelian group.

[4]We have that $a \mapsto (1,0)$, $a^{-1} \mapsto (-1,0)$, $b \mapsto (0,1)$ and $b^{-1} \mapsto (0,-1)$. Since $(\mathbb{Z} \times \mathbb{Z}, +)$ is abelian, the result follows.

## 6 Cayley Graphs

The type of hash function we will explore more thoroughly in this paper is Zémor's hash function, which is a type of Cayley hash function. We might wonder where the name "Cayley" in "Cayley hash function" comes from. The answer is that Cayley hash functions are based on a type of graph called Cayley graphs (named after Arthur Cayley), which is a presentation of a monoid according the following definition:

**Definition 6.1.** *Let $M$ be a monoid and $S$ a generating set of $M$. The Cayley graph $\Gamma = \Gamma(M, S)$ is a directed, colored graph constructed as follows:*

1. *Each element $g$ of $M$ is assigned a vertex; the vertex set $V(\Gamma)$ of $\Gamma$ is identified with $M$.*

2. *Each generator $s$ of $S$ is assigned a color $c_s$.*

3. *For any $g$ in $M$ and $s$ in $S$, the vertices corresponding to the elements $g$ and $gs$ are joined by a directed edge of colour $c_s$. Thus the edge set $E(\Gamma)$ consists of pairs of the form $(g, gs)$, with $s$ in $S$ providing the color.*

From the above definition, we can make the observation that each node of a Cayley graph generated by a set of cardinality $k$ will have an out-degree equal to $k$. The Cayley graph of a group need not be unique, and we will see an example of when two different generating sets for the same group produce completely different looking graphs. We will now have a look at the Cayley graphs for a few different monoids and groups.

**Example 6.1** (Cayley graph of $S_3$)**.** Below you can find the Cayley graph of the group $S_3$, generated by the elements (123) and (12). A red line represents a right action by (123), and a blue line represents a right action by (12).

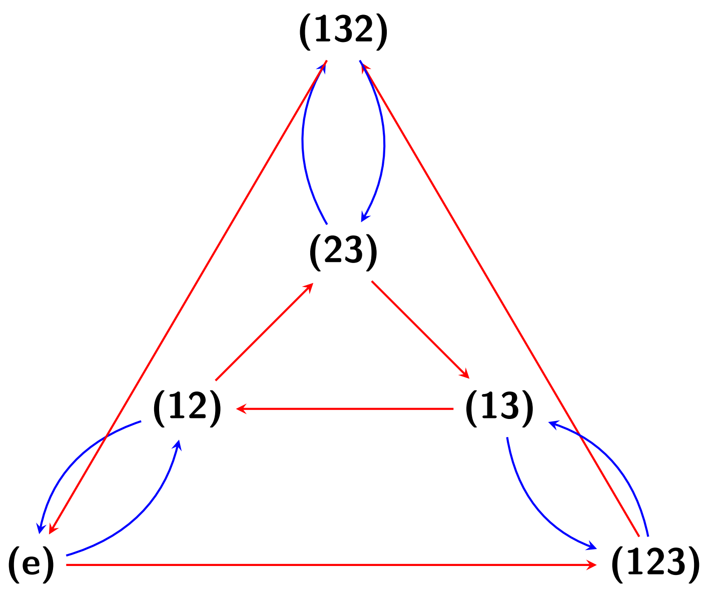


Figure 1: The Cayley graph of $S_3$ with generators (123) and (12).

A completely different looking graph of the same group, $S_3$, is generated if we instead choose generators (12) and (23). In the graph below, a blue line represents a left action by the element (23), and a red line represents a left action by the element (12). Since $S$ consists of transpositions there will always be an edge $(g, sg)$ whenever there is an edge $(sg, g)$, and therefore the arrow heads have been omitted.

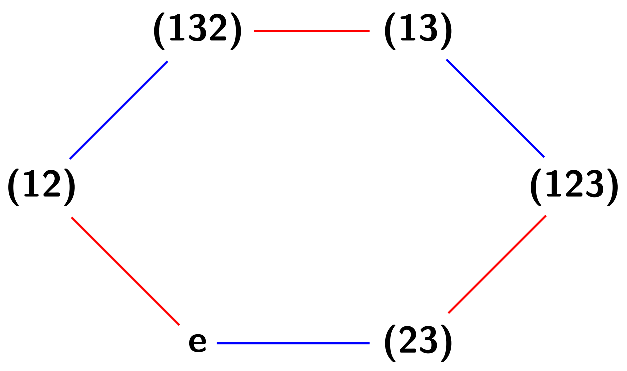


Figure 2: The Cayley graph of $S_3$ with generators (23) and (12).

As indicated previously, these graphs have entirely different properties as graphs regarding for example the distribution of cycle lengths.

**Example 6.2** (Cayley graph of $(\mathbb{N} \times \mathbb{N}, +)$)**.** We will now have a look at the Cayley graph of the monoid $(\mathbb{N} \times \mathbb{N}, +)$ of infinite order, with generators $(1, 0)$ and $(0, 1)$. In the graph below a blue line represents an action by $(1, 0)$ and a red line an action by $(0, 1)$.

$$\begin{tikzcd}
{} & {} & {} & \\
\mathbf{(0,2)} \arrow[u, dash, red] \arrow[r, blue] & \mathbf{(1,2)} \arrow[u, dash, red] \arrow[r, blue] & \mathbf{(2,2)} \arrow[u, dash, red] \arrow[r, dash, blue] & {} \\
\mathbf{(0,1)} \arrow[u, red] \arrow[r, blue] & \mathbf{(1,1)} \arrow[u, red] \arrow[r, blue] & \mathbf{(2,1)} \arrow[u, red] \arrow[r, dash, blue] & {} \\
\mathbf{(0,0)} \arrow[u, red] \arrow[r, blue] & \mathbf{(1,0)} \arrow[u, red] \arrow[r, blue] & \mathbf{(2,0)} \arrow[u, red] \arrow[r, dash, blue] & {}
\end{tikzcd}$$

Figure 3: The Cayley graph of $(\mathbb{N} \times \mathbb{N}, +)$.

**Example 6.3** (Cayley graph of the free group on two generators)**.** Below you can find an illustration of the free group generated by $S = \{a, b\}$ and its inverse set $S^{-1} = \{a^{-1}, b^{-1}\}$. The words created are the results of right actions on the identity, for example walking two steps to the right in the graph results in the word $aa$.

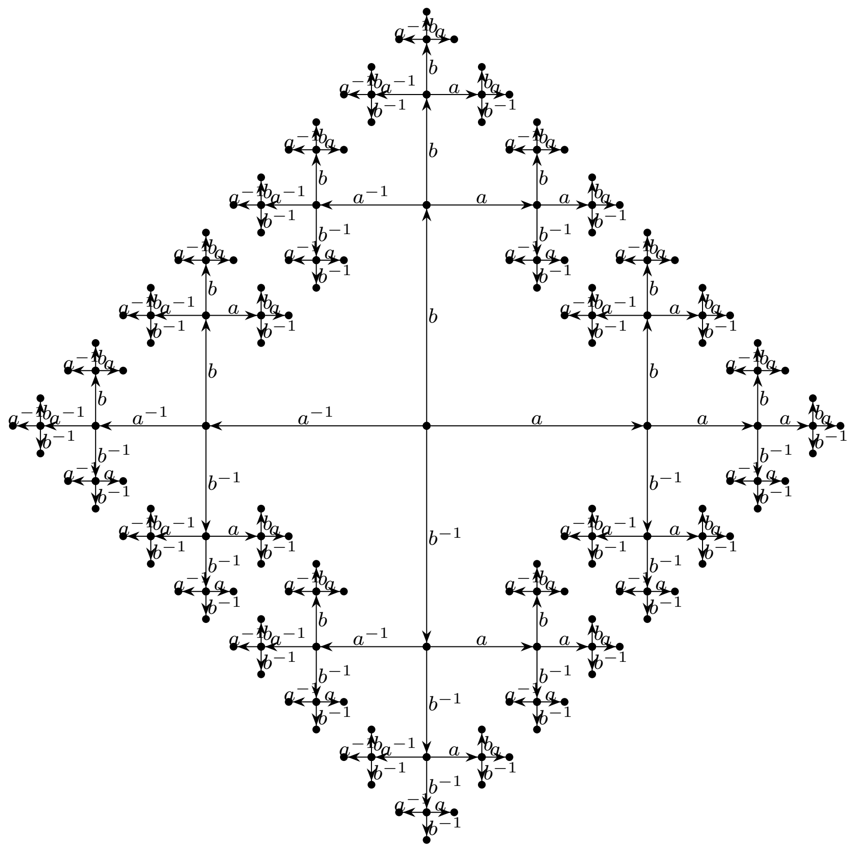


Figure 4: The Cayley graph of the free group on two generators. Image: https://tex.stackexchange.com/questions/222881/cayley-graph-of-free-group-in-tikz.

The broadly differing properties of Cayley graphs will play a big role ahead, as we start to look into Cayley hash functions.

## 7 Cayley Hash Functions

### 7.1 Definition of the Cayley Hash Function

We are now ready to shoulder one of the main events of this paper; getting acquainted with Cayley hash functions. We have previously thought of plaintexts as binary strings on $0, 1$, but we shall now consider them as strings of symbols from the alphabet $\{1, \ldots, k\}$ for $k \geq 2$. Any previous statements on plaintexts of binary strings is easily transferred back and forth to the situation we are now considering. With this in mind, based on a Cayley graph built from a subset $S = \{s_1, \dots s_k\}$ of a monoid $G$, we can build a Cayley hash function in the following way. Let $\sigma : \{1, \dots k\} \to S$ be an ordering and fix an initial element $g_0 \in G$. We then write the message to be hashed, $m$, as a string $m_1 m_2 \dots m_N$ s.t. $m_i \in \{1, \dots, k\}$. Our hash value is then the end of the walk we have created on our graph, with $g_0$ as starting point; $H(m) = g_0 \sigma(m_1) \dots \sigma(m_N)$. For two messages to have the same hash code, they must have the same start- and endpoints. We will assume $g_0$ is set to the identity, as the choice of starting point in fact does not affect the security of Cayley hash functions in the case where $G$ is a group (which is the case we will consider). Choosing the identity as starting point provides an important property of Cayley hash functions, which is their ability of being parallelized. In other words, they possess a *concatenation property*, meaning that the hash value for a concatenated message $xy$ simply is $H(xy) = H(x)H(y)$. This property makes it easier to handle big messages, as we can break them into smaller pieces and hash each piece [1]. The resistances of normal hash functions can be translated into group theoretic terms, in order to be fully compatible with Cayley hash functions.

**Group-theoretic problems of Cayley hash functions** *Let $G$ be a monoid and $S = \{s_1, \dots s_k\} \subset G$ a generating set. Further, let $L \in \mathbb{N}$ be small.*

1. **Factorization problem** Given an element $g \in G$, to find an "efficient" algorithm that returns a word $m_1 \dots m_l$ so that $l < L$, $m_i \in \{1, \dots, k\}$ and $\prod_{i=1}^{l} s_{m_i} = g$.

2. **Representation problem** To find an "efficient" algorithm which returns a word $m_1 \cdots m_l$ with $l < L, m_i \in \{1, \dots, k\}$ and $\prod_{i=1}^{l} s_{m_i} = 1$.

3. **Balance problem** To find an efficient algorithm that returns two different words $m_1 \dots m_l$ and $m'_1 \dots m'_{l'}$ with $l, l' < L$, $m_i, m'_i \in \{1, \dots, k\}$ that produce equal products in $G$, i.e. $\prod_{i=1}^{l} s_{m_i} = \prod_{i=1}^{l'} s_{m'_i}$.

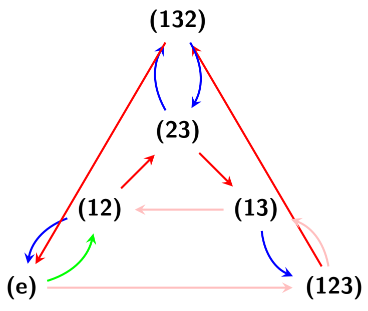
The above criterias give us an intuitive understanding of why the identity element is used as a starting point; given the group structure of $G$ we indeed have that $g_0 \prod_{i=1}^{l} s_{m_i} = g_0 \prod_{i=1}^{l'} s_{m'_i}$ if and only if $\prod_{i=1}^{l} s_{m_i} = \prod_{i=1}^{l'} s_{m'_i}$.

The Cayley hash function is pre-image resistant if and only if the factorization problem is hard, in other words it is equivalent to the problem of given an output

$y$ finding $x$ such that $H(x) = y$. This is in the case of Cayley hash functions the same as finding a message $m_1 \dots m_l$ such that $\sigma(m_1) \dots \sigma(m_l) = g$, i.e. finding the pre-image of $g$.

Further, the function is second pre-image resistant if and only if the representation problem is hard. Suppose that we have a message $m_1 \dots m_n$ and its hashed value $\prod_{i=1}^{n} s_{m_i}$. If we manage to find a message $m'_1 \cdots m'_j$ with $j < L$, $m'_i \in \{1, \dots, k\}$ such that $\prod_{k=1}^{j} s_{m'_k} = 1$, we will be able to constitute a new message by concatenating this message to the first, i.e. by constituting the message $m_1 \dots m_n m'_1 \dots m'_j$. Because of the previously mentioned parallelism property of Cayley hash functions, we have found a new message that maps to the same hash value as the original message $m_1 \dots m_n$.

Finally, the function is collision resistant if and only if the balance problem is hard. This is also quite straight forward if we note that $H(m) = H(m')$ occurs exactly when we find messages $m_1 \dots m_l$ and $m'_1 \dots m'_{l'}$ such that $\prod_{i=1}^{l} s_{m_i} = \prod_{i=1}^{l'} s_{m'_i}$. As an example of a collision we may consider the Cayley graph $S_3$. Starting at $e$, we may make our way to the element $(12)$ either by the path $(e) \to (123) \to (13) \to (12)$ or by the path $(e) \to (12)$, as seen in the picture below.



## 7.2 Girth of the Cayley Graph

In order to make sure a Cayley hash function fulfills the security requirements, some caution is needed when choosing the underlying Cayley graph. In this section, we shall put the balance problem in relation to the actual graph used for hashing. Translated into more graph theoretic terms, fulfilling the balance problem can be understood in terms of the Cayley graph used for hashing having a large *girth*.

**Definition 7.1.** *The* ***directed girth*** *of a graph is the largest integer $\delta$ such that given any two vertices $v$ and $w$, any pair of distinct directed paths joining the vertices will be such that one of those paths will have length $\delta$ or more. In case no two paths lead to the same endpoint, the girth is set to infinity.*

We will put this definition into the context of Cayley hash functions in a more concrete way by considering the proposition below, which can originally be found in [2]. In the following proposition by "message" we mean a plaintext.

**Proposition 7.2.** *Let $\delta$ be the directed girth of a Cayley graph $\Gamma$ built from the generating set $S = \{s_1, \dots, s_j\}$ of a monoid $M$. Further, let $H$ be a Cayley hash function based on $\Gamma$. Given a message*

$$m = m_1 \dots m_i m_{i+1} \dots m_{i+k} m_{i+k+1} \dots m_t,$$

*where $m_1, \dots m_t \in \{1, \dots, j\}$, if we replace $k$ of these letters with $h$ consecutive letters so that we obtain the below message*

$$m' = m_1 \dots m_i m'_{i+1} \dots m'_{i+h} m_{i+k+1} \dots m_t$$

*where $m'_i, \dots, m'_{i+h} \in \{1, \dots, j\}$ and $m'$ maps to the same hash value as $m$ by $H$, then $max(k, h) \geq \delta$.*

Let's try to gain a little more intuitive understanding of Proposition 7.2 by considering the above situation and supposing that we have the equality:

$$\begin{aligned}
H(m) &= H(m_1 \dots m_i m_{i+1} \dots m_{i+k} m_{i+k+1} \dots m_t) \\
&= H(m_1 \dots m_i m'_{i+1} \dots m'_{i+h} m_{i+k+1} \dots m_t) \\
&= H(m').
\end{aligned}$$

This means that there are two paths to $H(m)$; one path is obtained by hashing

$$m_1 \dots m_i m_{i+1} \dots m_{i+k} m_{i+k+1} \dots m_t$$

and the other by hashing

$$m_1 \dots m_i m'_{i+1} \dots m'_{i+h} m_{i+k+1} \dots m_t.$$

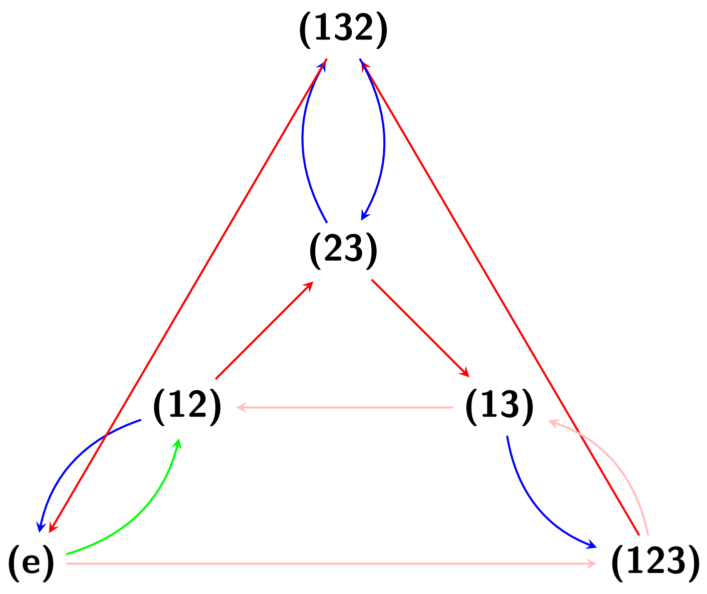
Using the parallelism property, this means that these two paths part ways after $H(m_i)$, and reunite in $H(m_{i+k})$. Since the girth of the graph is $\delta$, we must have that at least one of $m_{i+1} \dots m_{i+k}$ and $m'_{i+1} \dots m'_{i+h}$ has a length that is at least $\delta$, i.e. $\max(k, h) \geq \delta$.

The reason Cayley graphs used for hashing should have large girths, is that the hash functions used over these graphs do not produce any short relations. This means that the probability that two paths lead to the same endpoint is small, minimizing the number of collisions. Intuitively, this can be understood as given a path, we are forced to make several "correct mistakes" in order to reach the same destination by a different path. An example of a graph that for this reason is *not* suitable for hashing is any graph representing an abelian group; here any unique (unordered) set of elements will simply produce identical hash values as order is irrelevant! A similar example of a Cayley graph with small girth is the graph of the abelian monoid $\mathbb{N} \times \mathbb{N}$, in which we find several short-length cycles. Here, for example, I have colored two paths leading from and to the same nodes in green and pink.

$$\begin{tikzcd}
{} & {} & {} & \\
\mathbf{(0,2)} \arrow[u, dash, red] \arrow[r, blue] & \mathbf{(1,2)} \arrow[u, dash, red] \arrow[r, blue] & \mathbf{(2,2)} \arrow[u, dash, red] \arrow[r, dash, blue] & {} \\
\mathbf{(0,1)} \arrow[u, red] \arrow[r, pink] & \mathbf{(1,1)} \arrow[u, red] \arrow[r, blue] & \mathbf{(2,1)} \arrow[u, red] \arrow[r, dash, blue] & {} \\
\mathbf{(0,0)} \arrow[u, pink] \arrow[r, green] & \mathbf{(1,0)} \arrow[u, green] \arrow[r, blue] & \mathbf{(2,0)} \arrow[u, red] \arrow[r, dash, blue] & {}
\end{tikzcd}$$

Given the pink path we only need to make two "correct mistakes" in order to obtain a different path with the same end point. In other words, it is actually fairly easy to find two messages $x \neq y$ such that $H(x) = H(y)$, and hence the balance problem in fact seems to be rather easy to solve.

We will now revisit the Cayley graph of $S_3$ with generators (123) and (12), which has a larger girth than the previous graph. Given, for example, the green path starting at $e$ and going to (12), we have to make 3 "correct mistakes" in order to end up at the same end vertex, i.e. to obtain the identical hash value as our initial value. However, this graph is also deficient in terms of forming a proper base for hashing, since we only need to check a few possible combinations before we reach a collision.



Collisions are in fact not possible at all in some cases; consider a graph with infinite girth such as the free group on two generators, whose graph we saw previously. Since a free monoid or group contains no relations, producing two unique messages whose group products are equal becomes impossible. Judging from this fact, the Cayley graph of a free monoid might seem like a perfectly fit choice for a hash function. However, as hash functions must take on values from a finite set, we can not simply use an infinite group for hashing. One idea which we will explore further a head, however, is to use a free monoid whose elements are reduced over a finite field $F_p$, for some large prime $p$.

## 7.3 Expanding Properties

Preferable in cryptographic hashing, is for the hashed values to appear random. A perceived randomness can be achieved when the graph chosen for hashing is an *expander graph*. In this section I will give a brief overview of the properties of such graphs, and why they are interesting to us from a cryptographic view point.

For the sake of intuition, let us imagine a graph to model a social network, where each vertex is a person and each edge implies that two people gossip. If our graph is an expander graph, rumours spread very quickly in this social network [5]. Put in slightly more formal terms, expander graphs are finite graphs that possess two properties that might appear contradictory, as they are simultaneously sparse and well-connected. Further, every subset $U$ of vertices of an expander graph $G$ is connected to "many" vertices in $\overline{U}$ (where $\overline{U} = \{v|v \in V(G) \setminus U\}$, where $V(G)$ is the vertex set of $G$). As a consequence, expander graphs have a low diameter [5] and high chromatic number.[6] Now, let $G = (V, E)$ be an undirected graph, with $|V| = n$. The set of edges between two sets of vertices, $U, W \in V$, is denoted by $E(U, W) = \{(u, w)|(u, w) \in E, u \in U, w \in W\}$. The *edge boundary* of $U \subset V$ is defined as $\delta U = E(U, \overline{U})$. From the edge boundary, we can define the following:

**Definition 7.3.** *The expansion constant of $G$ is defined by*

$$h(G) = \min_{\substack{0<|U|\leq \frac{n}{2} \\ U \subset V}} \frac{|\delta U|}{|U|}.$$

$h(G)$ is strictly positive if and only if $G$ is a connected graph. A small yet positive value of $h(G)$ implies that the graph has a "bottleneck"; that there are two sets of vertices in the graph that only have a few edges connecting them. A high positive value on the other hand, indicates that any two sets in the graphs have "many" edges between them.

When talking about expander graphs, one usually considers families of graphs rather than just one graph. A family of expander graphs can be constructed in accordance to the following:

**Definition 7.4.** *A family of expander graphs $\{G_i\}_{i\in\mathbb{N}}$ is a sequence of $d-$regular graphs $\{G_i\}_{i\in\mathbb{N}}$ of size increasing with $i$, such that there exists $\epsilon > 0$ such that $h(G_i) \geq \epsilon$ for all $i$.*

What the above tells us in an intuitive sense, is that the graphs in consideration are arbitrarily large but are all $d-$regular. Hence, they become more and more sparse. Simultaneously, there exists some positive constant $\epsilon$ such that each

[5]The diameter of a graph is the largest distance between any two of its vertices

[6]The chromatic number of a graph is the minimum number of colors needed to produce a proper coloring of the graph.

graph has an expander constant which is larger. We will now have a look at an example from [6] of a family of expanders.

**Example 7.1** (The random graph)**.** Let $G_n$ be a random $d-$regular graph with $n$ vertices. We can construct a random $d$-regular graph by connecting each vertex to $d$ randomly chosen vertices. Let $U$ be a subset of vertices of $G_n$ such that $|U| \leq \frac{n}{2}$. A vertex of $U$ will then be connected to $\approx d\frac{|\overline{U}|}{n}$ vertices in $\overline{U}$ (since the probability that a vertex lies in $\overline{U}$, and hence the proportion of vertices in $\overline{U}$, is $\frac{|\overline{U}|}{n}$). There are $|U|$ vertices in $U$, and hence the number of edges between $U$ and $\overline{U}$ should be approximately $|U| \cdot d\frac{|\overline{U}|}{n}$. We have:

$$|\delta U| \approx |U| \cdot d\frac{|\overline{U}|}{n}$$

$$\iff$$

$$\frac{|\delta U|}{|U|} \approx d\frac{|\overline{U}|}{n}$$

$$\implies$$

$$\underset{\substack{0<|U|\leq\frac{n}{2} \\ U\subset V}}{min} \frac{|\delta U|}{|U|} \approx \underset{\substack{0<|U|\leq\frac{n}{2} \\ U\subset V}}{min} d\frac{|\overline{U}|}{n}.$$

We have that $d\frac{|\overline{U}|}{n}$ is minimal when $|\overline{U}|$ is minimal, which occurs when $|U|$ is maximal, i.e. when we have $|U| = \frac{n}{2}$. This means that:

$$|\overline{U}| = n - |U| = n - \frac{n}{2} = \frac{n}{2}$$

which gives us:

$$\underset{\substack{0<|U|\leq\frac{n}{2} \\ U\subset V}}{min} \frac{|\delta U|}{|U|} \approx \frac{d\frac{n}{2}}{n}$$

$$\Longrightarrow$$

$$h(G_n) \approx \frac{d}{2}.$$

We see that $h(G_n)$ does not depend on $n$, and hence a family of $d-$regular random graphs is an expander family.

In order to put the notion of expander graphs in more close relation to Cayley hash functions and Cayley graphs, worth mentioning is that we in [5] learn that an infinite family of groups $\{G_n\}$ can be made into a family of expanders if there is some constant $k$ and a generating set $S_n$ where $|S_n| = k$ for each $G_n$, so that the family $\{\Gamma_{G_n,S_n}\}$ is a family of expanders. Even more interesting in regard to the subject of hashing with matrices which we will be covering in the coming chapters, is that the special linear groups $SL(d, \mathbb{F}_{p^m})$ for any $d \geq 2$, $m \geq 1$ and prime $p$ can be made into a family of expanders as $p \to \infty$ according to [5] (the proof is however too extensive to cover here).

# 8 Cayley Hash Functions Using Matrices

## 8.1 Zémor's Hash Function

In this chapter, we will explore different versions of the Cayley hash function using matrices. The idea behind this type of hash function is to use pairs of generators $A$ and $B$ of some monoid $M$, such that the Cayley graph of $M$ generated by $A$ and $B$ is an expander [1]. To ensure that the requirements for cryptographic hash functions are fulfilled, one must assure that the graph generated by $A$ and $B$ has a large girth, in order to avoid short relations (as seen in the previous chapter). In 1991 Zémor introduced the idea of hashing with matrices by proposing a Cayley hash function in which $A$ and $B$ are $2 \times 2$ matrices. Matrices $A$ and $B$ are chosen so that they generate a free monoid over $SL(2, \mathbb{Z})$, to hash the 0 and 1 bit respectively. The hashing of a bit string is then matrix multiplication, so that for example the message 00010 in bits would be hashed to the matrix product $AAABA = A^3BA$. The entries of the resulting matrix are reduced modulo some large prime $p$, and hence we obtain a matrix of $SL(2, \mathbb{F}_p)$, where $\mathbb{F}_p$ is a field on $p$ elements and $p$ is a large prime (with "large" meaning about 150 bits). This way, one obtains a lower bound on the length of collisions. Since multiplying two matrices only accounts for 4 additions, hashing an $n$-bit text requires the reasonably small amount of $4n$ additions.

We will in particular be looking at Zémors first proposal of such matrices, $A(1)$

and $B(1)$:

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

$A(1)$ and $B(1)$ as generators of the group $SL(2, \mathbb{F}_p)$ have a Cayley graph that is an expander graph [4]. Actually, these matrices generate the *entire* monoid $SL(2, \mathbb{Z}_+)$, and together with their inverses:

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

they generate the group $SL(2, \mathbb{Z})$. This group is not free, but since only positive powers are needed for hashing, it would be sufficient to find a *free monoid* in $SL(2, \mathbb{Z}_+)$.

We will also familiarize ourselves with an improvement of Zémor's first proposal, in which Bromberg suggests matrices $A(n)$ and $B(n)$ for $n \geq 2$:

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

The above matrices as generators of a submonoid of $SL(2, \mathbb{F}_p)$ also has a Cayley graph which forms an expander according to [4].

We must, however, make sure that $A(1)$ and $B(1)$, as well as $A(n)$ and $B(n)$ for $n \geq 2$, actually do generate a free monoid in $SL(2, \mathbb{Z})$.

## 8.2 $A(n)$ and $B(n)$ Generate a Free Monoid

We claim that coinciding products in $A(1)$ and $B(1)$ implies equal words or concatenations of $A(1)$ and $B(1)$. Our first stepping stone is to show the below lemma.

**Lemma 8.1.** *If two products of $A(1)$ and $B(1)$ are equal, i.e. if:*

$$X_1 \dots X_m = Y_1 \dots Y_n,$$

*where $X_i, Y_i \in \{A(1), B(1)\}$, then $X_1 = Y_1$.*

*Proof.* We will assume the contrary, i.e. we will assume that $X_1 \neq Y_1$, but that $X_1 \dots X_m = Y_1 \dots Y_n$. This means we have an equality in accordance to the following:

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}.$$

Multiplying the matrices above gives the following equation:

$$\begin{pmatrix} a+c & b+d \\ c & d \end{pmatrix} = \begin{pmatrix} a' & b' \\ a'+c' & b'+d'. \end{pmatrix}. \tag{1}$$

This equality implies the following system of equations:

$$\begin{aligned} a+c &= a' && (2) \\ b+d &= b' && (3) \\ c &= a'+c' && (4) \\ d &= b'+d'. && (5) \end{aligned}$$

Subtracting (2) from (4) gives us:

$$\begin{gathered} c-(a+c) = a'+c'-a' \\ \Longleftrightarrow \\ -a = c'. \end{gathered}$$

Since no negative integer can occur in the matrix, this implies that $-a = c' = 0$. In a similar manner we get by subtracting (3) from (5):

$$\begin{gathered} d-(b+d) = b'+d'-b' \\ \Longleftrightarrow \\ -b = d' \end{gathered}$$

which has the only non-negative solution $-b = d' = 0$. These equalities give us for (1):

$$\begin{pmatrix} c & d \\ c & d \end{pmatrix} = \begin{pmatrix} a' & b' \\ a' & b' \end{pmatrix}.$$

But these matrices can not appear as products of $A(1)$ and $B(1)$, as they clearly have determinants equal to 0! The reason for this can be realised through a property of multiplication of determinants from linear algebra, which says that $det(AB) = det(A) \cdot det(B)$. Since all factors appearing in the products have determinants equal to 1, the resulting matrix can not possibly have a determinant equal to 0. This leads to the conclusion that $X_1 = Y_1$. □

We proceed by showing that each unique word built of $A(1)$ and $B(1)$ is uniquely connected to its matrix product, and hence that $A(1)$ and $B(1)$ generate a free monoid.

**Lemma 8.2.** *If two products of $A(1)$ and $B(1)$ are equal, i.e. if:*

$$X_1 \dots X_m = Y_1 \dots Y_n$$

*where $X_i, Y_i \in \{A(1), B(1)\}$, then $m = n$ and $X_i = Y_i$ for all $i$.*

*Proof.* We prove the lemma by induction on $k = min(m, n)$. Our base case is $k = 0$, which is the identity matrix. This means that we need to show that there is no equality:

$$X_1 \cdots X_j = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

where $X_i \in \{A(1), B(1)\}$ and $j > 0$. Now, let's assume that the above equation holds and let:

$$M = X_1 \cdots X_{j-1} = \begin{pmatrix} x & y \\ z & t \end{pmatrix}.$$

Then, multiplying $M$ by $A(1)$ or $B(1)$ on the right results in the following products:

$$M_1 = \begin{pmatrix} x & y \\ z & t \end{pmatrix} A(1) = \begin{pmatrix} x & y \\ z & t \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} x & x+y \\ z & z+t \end{pmatrix}$$

$$M_2 = \begin{pmatrix} x & y \\ z & t \end{pmatrix} B(1) = \begin{pmatrix} x & y \\ z & t \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} x+y & y \\ z+t & t \end{pmatrix}$$

of which (at least) one must be the identity, according to our assumption. Since $M$ itself is a product of $A(1)$:s and $B(1)$:s, its entries are non-negative, and hence $M_1$ and $M_2$ must also have non-negative entries. Proceeding, we assume that $M_1$ equals the identity which means that we must have that:

$$\begin{aligned} x + y &= 0 \\ z &= 0. \end{aligned}$$

This means that $x = -y$, forcing the equality $x = -y = 0$ (since $M_1$ has non-negative entries). This results in the following matrix:

$$\begin{pmatrix} 0 & 0 \\ 0 & t \end{pmatrix}$$

which clearly has determinant 0. This is however impossible, since (again) by the rule of determinants $det(AB) = det(A) \cdot det(B)$, and $M_1$ is a product of $A(1)$:s and $B(1)$:s with determinant 1. Hence, $M_1$ can not equal the identity. Assuming that $M_2$ equals the identity gives us:

$$\begin{aligned} z + t &= 0 \\ y &= 0 \end{aligned}$$

which means that $z = -t$, with the only possibility that $z = -t = 0$. This results in the following matrix:

$$\begin{pmatrix} x & 0 \\ 0 & 0 \end{pmatrix}.$$

Applying the exact same argument as for $M_1$, we realize that obtaining this matrix as a product of $A(1)$:s and $B(1)$:s is impossible. Hence, $M_2$ can not equal the identity.

We now suppose the lemma to be true for all $k \leq k_0$, and try to prove it for $k = k_0 + 1$. Hence, suppose we have the following equality:

$$X_1 \dots X_m = Y_1 \dots Y_n.$$

By lemma 8.1, we know that $X_1 = Y_1$, and since $A(1)$ and $B(1)$ are invertible we obtain:

$$X_2 \dots X_m = Y_2 \dots Y_n.$$

The above products have lengths $m-1$ and $n-1$, and we have that $min(m-1, n-1) = k_0$. By the induction hypothesis we have that our lemma holds for $k_0$, and hence $m = n$ and $X_i = Y_i$ for all $i$.

$\square$

**Corollary 8.2.1.** *Every specific concatenation of the matrices $A(1)$ and $B(1)$ results in a unique product under matrix multiplication. This implies that the matrices $A(1)$ and $B(1)$ generate a free monoid.*

**Lemma 8.3.** *We have that:*

$$\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^m = A(1)^m$$

*and also that:*

$$\begin{pmatrix} 1 & 0 \\ m & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^m = B(1)^m.$$

*Proof.* We proceed by induction on $m$. For the base case $m = 0$ we have that:

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and

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

We assume the lemma holds for $m, n \leq k$, and try to prove it holds for $k+1$. Indeed, for the $(k+1)$-th step we get:

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{k+1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & k+1 \\ 0 & 1 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^{k+1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ k+1 & 1 \end{pmatrix}$$

which shows that the lemma holds.

□

**Corollary 8.3.1.** *The monoid generated by $A(m)$ and $B(n)$, for arbitrary $n, m \in \mathbb{Z}_+$ is free.*

*Proof.* Since we have that $A(m) = A(1)^m$ and $B(n) = B(1)^n$, any product of $A(m)$ and $B(n)$ can be rewritten as a product of $A(1)$ and $B(1)$. This means that this case is analogous to the previous one, i.e. that each unique concatenation of $A(m)$ and $B(n)$ is connected to a unique product. Hence, $A(m)$ and $B(n)$ generate a free monoid. □

We have shown that $A(m)$ and $B(n)$ for $m, n \in \mathbb{N}$ generate a free monoid. In particular, this means that these elements are suitable generators for Cayley hash functions.

# 9 The Lifting Attack

## 9.1 An Attack Using Euclid's Algorithm

The hash function introduced by Zémor with generators $A(1)$ and $B(1)$ was broken by Tillich and Zémor in 1993, as they were able to produce the most efficient documented attack against the Zémor hash function yet, known as the *lifting attack*. The fact that $A(1)$ and $B(1)$ generate the entire $SL(2, \mathbb{Z}_+)$ made it possible to attack the hash function by lifting the representation problem to a monoid in which it is easier to solve. By using the simple Euclid's algorithm, the hash value could then simply be factorized into a product of $A(1)$ and $B(1)$. In this chapter we will get acquainted with the main ideas of this attack, and also look at an example.

The goal of the lifting attack is to produce a collision with a given message, which is done by finding a matrix $U$ in $SL(2, \mathbb{Z}_+)$ that reduces modulo $p$ to the identity (see section 7.1). Given a hash value $M$ which factors into two products $P$ and $Q$ such that $M = PQ$, an adversary can create a new message by concatenating $P$, $U$ and $Q$ into $PUQ$, which will collide with the original message $M$ [2]. Before we look at the attack itself, let us briefly pause to make a group theoretic observation. We are looking for some matrix $U$ which reduces modulo $p$ to the identity. We also have that the balance problem consists of finding two strings or messages, such that they have coinciding products in $SL(2, \mathbb{F}_p)$, in other words so that we have:

$$s_1 s_2 \dots s_n = s'_1 s'_2 \dots s'_j \tag{6}$$

where $s_1, s_2, \dots, s_n, s'_1, s'_2, \dots, s'_j \in S$. Because of the group structure of $SL(2, \mathbb{F}_p)$, this implies the following relation:

$$s_1 s_2 \dots s_n {s'_j}^{-1} \dots {s'_2}^{-1} {s'_1}^{1} = 1. \tag{7}$$

Finding a factorization of (7) corresponds to breaking the representation problem, and hence we see that the group structure of $SL(2, \mathbb{F}_p)$ provides an equality between the representation problem and the balance problem (see section 7.1) [3]. We might wonder if we can simply use the trivial factorization for (7), since we have that $s^{|G|} = 1$ for any element $s$ of a group $G$. But if we choose a group of very large cardinality, say $|G| = 2^{500}$, trivial factorization becomes impossible since no message is as long as $2^{500}$ bits [3].

In order to break the representation problem, an adversary looks for a matrix on the following form:

$$U = \begin{pmatrix} 1+k_1p & k_2p \\ k_3p & 1+k_4p \end{pmatrix}$$

in $SL(2, \mathbb{Z}_+)$, which clearly reduces modulo $p$ to the identity. Important to note here is the $+$-subscript and the fact that $k_i$ for $i = 1, \ldots, 4$ must be positive. In other words, $U$ must have non-negative entries which follows from the fact that a product of $A(1)$ and $B(1)$ can not have negative entries. Proceeding, since $U$ is a matrix of $SL(2, \mathbb{Z}_+)$, its determinant has to be 1. This gives us the following equation:

$$(1 + k_1p)(1 + k_4p) - k_2k_3p^2 = 1,$$

which can be simplified:

$$\begin{gathered} (k_1 + k_4)p + (k_1k_4 - k_2k_3)p^2 = 0 \\ \Longleftrightarrow \\ (k_1 + k_4) + (k_1k_4 - k_2k_3)p = 0. \end{gathered}$$

The adversary then looks for a solution for $(k_1, k_2, k_3, k_4)$ such that $k_1+k_4 = cp$, where $c$ is some small integer. She then chooses a random prime $p'$ of the same order as $p$, and let $k_3 = p'$. Substituting for $k_1 + k_4$ and $k_4$ into the equation above we receive:

$$\begin{gathered} cp + (k_1(cp - k_1) - k_2p')p = 0 \\ \Longleftrightarrow \\ c + k_1(cp - k_1) - k_2p' = 0 \\ \Longleftrightarrow \\ c + k_1(cp - k_1) = k_2p' \\ \Longleftrightarrow \\ c + k_1cp - k_1^2 = k_2p'. \end{gathered}$$

This equals to solving:

$$k_1^2 - cpk_1 - c = 0 \text{ (mod } p').$$

By the usual formula for solving quadratic equations, this means that the discriminant $c^2p^2 + 4c$ must be a quadratic residue[7] modulo $p'$. After a small number of random tries, the adversary should be able to find $c, p'$ such that:

$$\begin{cases} k_1 = \frac{cp+\sqrt{c^2p^2+4c}}{2} \text{ (computation done mod } p'\text{)}, \\ k_2 = \frac{c+cpk_1-k_1^2}{p'} \text{ (computation done over the integers)}, \\ k_3 = p', \\ k_4 = cp - k_1 \end{cases}$$

where $\sqrt{x}$ denotes a positive representation of a square root of $x$ modulo $p'$, if $x$ is a quadratic residue modulo $p'$.

Once $k_1, k_2, k_3$ and $k_4$ are found, the remaining problem is to factor the obtained matrix $U$, in other words the factorization problem is still left to solve. This problem is significantly simplified by the possibility of using the Euclidean algorithm for factoring, of which we will give a general sketch and an example (for details see [2]). Let

$$U = \begin{pmatrix} \alpha & \beta \\ \gamma & \rho \end{pmatrix}$$

be the obtained matrix which reduces to the identity modulo $p$. We will consider the case where $\alpha > \beta$, and we will perform Euclid's algorithm on $\alpha, \beta$. In our initial step of the algorithm, we get:

$$\alpha = \beta q_1 + r_1$$
$$\beta = r_1 q_2 + r_2.$$

The above can be expressed in matrix form the following way:

$$\begin{pmatrix} \alpha & \beta \end{pmatrix} = \begin{pmatrix} r_1 & r_2 \end{pmatrix} \begin{pmatrix} 1 & q_2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ q_1 & 1 \end{pmatrix}.$$

We then proceed by repeating the algorithm on $(r_1, r_2)$. To generalize the procedure, the $i$-th step of the Euclidean algorithm is $r_{i-2} = q_{i-1}r_{i-1} + r_i$, and can be expressed in matrix form as:

[7]An integer $q$ is called a quadratic residue modulo $n$ if it is congruent to a square modulo $n$, i.e. if there is an integer $x$ such that $x^2 \equiv q \pmod{n}$.

$$\begin{pmatrix} r_{i-2} & r_{i-1} \end{pmatrix} = \begin{pmatrix} r_i & r_{i+1} \end{pmatrix} \begin{pmatrix} 1 & q_i \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ q_{i-1} & 1 \end{pmatrix}.$$

We apply the Euclidean algorithm on $(r_i, r_{i+1})$ until we reach the standard base vector, $(r_i, r_{i+1}) = (1, 0)$. If the number of steps in Euclid's algorithm $n$ is even, the above gives us the following factorization of $U$ :

$$U = U' \begin{pmatrix} 1 & q_n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & q_{n-1} \\ 0 & 1 \end{pmatrix} \dots \begin{pmatrix} 1 & 0 \\ q_1 & 1 \end{pmatrix}$$

where the first row of $U'$ is $(1, 0)$ and the second row is $(a, 1)$, where $a$ is a positive integer. We can then find $U'$ by calculating:

$$U' = U \begin{pmatrix} 1 & 0 \\ q_1 & 1 \end{pmatrix}^{-1} \dots \begin{pmatrix} 1 & q_n \\ 0 & 1 \end{pmatrix}^{-1}.$$

In order to obtain a factorization into factors $A(1)$ and $B(1)$, we use the result of corollary 8.2.2 which tells us that:

$$\begin{pmatrix} 1 & q \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^q$$

and

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

Putting it all together, if the number of steps in the algorithm is even, we will have found a factorization on the form:

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \rho \end{pmatrix} = A(1)^{q_n} B(1)^{q_{n-1}} \dots A(1)^{q_2} B(1)^{q_1}$$

where $q_1, q_2, \dots, q_n$ are the quotients in Euclid's algorithm.

**Example 9.1** (The lifting attack)**.** We will go through every step of a slightly altered example originally found in [2], illustrating how Euclid's algorithm should be used. Let $p = 5$, and we choose $c = 1$. We now need to find $p'$ such that $c^2p^2+4c$ is a quadratic residue modulo $p'$, in other words $1\cdot 25+4\cdot 1 = 29$ must be a quadratic residue modulo $p'$. Taking $p' = 7$ we have that $29 \equiv 1 \equiv 6^2 \bmod 7$. This gives us the following equation:

$$k_1^2 - 5k_1 - 1 = 0 \text{ mod } 7,$$

with solution:

$$k_1 = \frac{5+\sqrt{29}}{2} = 3,$$

which gives us the full solution

$$\begin{cases} k_1 = \frac{5+\sqrt{5^2+4}}{2} = 3 \text{ (computation done mod 7)}, \\ k_2 = \frac{1+5\cdot 3-3^2}{7} = 1 \text{ (computation done over the integers)}, \\ k_3 = p' = 7, \\ k_4 = 5-3 = 2. \end{cases}$$

Having solved the above, we obtain the following matrix:

$$U = \begin{pmatrix} 1+k_1p & k_2p \\ k_3p & 1+k_4p \end{pmatrix} = \begin{pmatrix} 16 & 5 \\ 35 & 11 \end{pmatrix}.$$

Left to do for a forger now is to factorize the above with the help of Euclid's algorithm, which is performed on $(16, 5)$:

$$\begin{aligned} 16 &= 5\cdot 3+1 \\ 5 &= 1\cdot 5+0. \end{aligned}$$

The above can be rewritten in matrix form as:

$$\begin{pmatrix} 16 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}.$$

We hence have, for the factorization of $U$:

$$U = U' \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$$

$$\iff$$

$$U' = U \begin{pmatrix} 1 & 0 \\ -3 & 1 \end{pmatrix} \begin{pmatrix} 1 & -5 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}.$$

Putting this all together, we obtain:

$$\begin{pmatrix} 16 & 5 \\ 35 & 11 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}.$$

Using corollary 8.2.2, we obtain a full factorization of $U$ as a product of $A(1)$ and $B(1)$:

$$\begin{pmatrix} 16 & 5 \\ 35 & 11 \end{pmatrix} = B(1)^2 A(1)^5 B(1)^3 \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mod 5.$$

## 9.2 Avoiding the Attack

As Zémor's first proposal of matrices $A(1)$ and $B(1)$ were proven to be insufficient in fulfilling the security requirements, Bromberg suggests matrices on the form:

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

where $n \geq 2$. The reason for this is that while the lifting attack against the hash function with generators $A(1)$ and $B(1)$ becomes possible since $A(1)$ and $B(1)$ generate the whole monoid $SL(2, \mathbb{Z}_+)$, this is, however, not the case for all generators $A(n)$ and $B(n)$. In the following section I shall sketch an intuitive explanation of why the hash function on these generators stand secure against lifting attacks (as far as is known).

Bromberg proposes choosing generators $A(n)$ and $B(n)$ with the property that the probability that a random matrix in $SL(2, \mathbb{Z}_+)$ can be decomposed into a product of $A(n)$ and $B(n)$, is small. In other words, we wish to find $A(n)$ and $B(n)$ such that they generate a sufficiently sparse submonoid of $SL(2, \mathbb{Z}_+)$. The sets

$$\{A(2) = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, B(2) = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}\} \text{ and } \{A(3) = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}, B(3) = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}\}$$

fulfill this criteria [4]. Looking more closely at $A(2)$ and $B(2)$, the subgroup of $SL(2, \mathbb{Z})$ generated by $A(2)$ and $B(2)$ and their inverses consists of all matrices on the form

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

where every $m_i$ is an arbitrary integer [4]. Bromberg states however, that a generic matrix of the above form does not belong to the *monoid* generated by the $A(2)$ and $B(2)$. This can be explained the following way; $A(2)$ and $B(2)$ generate (together with their inverses) *a free group*, whereupon each unique concatenation of the elements is linked to a unique product. Hence, the number of elements generated by $A(2)$ and $B(2)$ together with their inverses of length $m \geq 2$ is $4 \cdot 3^{m-1}$ (there are four choices for the first letter, and then 3 choices for each following letter since an element can not be followed by its inverse), while the number if elements represented by *positive words* of length $m \geq 2$ is much smaller; only $2^m$ (there are two choices for each letter). Hence, the number of matrices on the above form that are elements of the monoid, i.e. the words on *positive letters*, is negligible [4]. Hence, we realize that the situation is very different from the one with $A(1)$ and $B(1)$, where *any* matrix of $SL(2, \mathbb{Z}_+)$ can be factorized into a product of $A(1)$ and $B(1)$.

The lifting attack can, however, still produce collisions in the *group* generated by $A(2)$ and $B(2)$, but it becomes futile to use on hash functions based on $A(2)$ and $B(2)$ since then only positive powers of $A(2)$ and $B(2)$ are used. According to Bromberg, a result for the group generated by $A(3)$ and $B(3)$ corresponding to that of $A(2)$ and $B(2)$ has not yet been reached. Therefore, there is no efficient algorithm capable of creating relations between words neither in the group on $A(3)$ and $B(3)$ nor their monoid [4].

## 10 A Word on the Girth of Cayley Graphs

As articulated in their requirements, of paramount importance is the property of collision resistance of Cayley hash functions. Since $A(n)$ and $B(n)$ generate a free monoid in $SL(2,\mathbb{Z}_+)$ which then are reduced modulo $p$ to obtain elements of $SL(2,\mathbb{F}_p)$, there is a lower bound for when collisions can occur. More explicitly, at least one entry in at least one of the products must be at least $p$ for collisions to be possible. For the sake of security, one would prefer that collisions are possible only for long words - that way one would have the advantage of plenty of choices for a message before a collision can take place. One way of ensuring collision resistance is to examine how many letters messages must contain before a collision can occur, in other words how large $p$ should be on the basis of how fast the elements of the matrices grow. In fact, this speed depends on the structure of the message. According to Bromberg, entries in a matrix product of length $k$ grow faster when the product is alternating in positive powers of $A(n)$ and $B(n)$, i.e. when the product is on the form $A(n)B(n)A(n)\dots$. The precise statement is given in the following proposition found in [4]:

**Proposition 10.1.** *Let $w_m(a,b)$ be an arbitrary positive word of even length $m$, and let $W_m = w_m(A(n),B(n))$ with $m \geq 2$. Let $C_m = (A(n)\cdot B(n))^{\frac{m}{2}}$, i.e. the alternating product of $A(n)$ and $B(n)$ of length $m$. Then there is a row in $C_m$ such that:*

*a) its sum is at least as large as any row sum of $W_m$,*

*b) its largest entry is at least as large as the largest entry of $W_m$.*

(Note that Bromberg arrives at a slightly different result in [4], where she claims that the row sum of *any* row of $C_m$ is at least as large as *any* row of $W_m$. This has been altered here, since it is in fact not true for $m = 2$.)

Before giving a proof of the proposition, I will give a brief intuitive explanation. First of all, multiplying any matrix $X$ by $A(n)$ on the right equals to adding the first column of $X$ multiplied by $n$ to the second. Further, multiplying $X$ by $B(n)$ on the right equals to adding the second column in $X$ multiplied by $n$ to the first. Hence, when building words of $A(n)$ and $B(n)$, the elements change independently row-wise; if we would multiply solely by $A(n)$ on the right for example, only the elements of the second column would be increasing as multiples of the first column would be added to it. Hence no addition between the rows occur. With this rather intuitive approach in mind, we are now ready to give a proper proof.

*Proof.* We shall proceed by induction on the length of words, $m = 2k$, to show that an alternating sequence of $A(n)$ and $B(n)$ with $n \geq 2$, results in the largest row sum and element. Our base case is $m = 2$, resulting in the following cases:

$$B(n)B(n) = \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2n & 1 \end{pmatrix}$$

$$A(n)A(n) = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2n \\ 0 & 1 \end{pmatrix}$$

$$A(n)B(n) = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} = \begin{pmatrix} n^2+1 & n \\ n & 1 \end{pmatrix}$$

$$B(n)A(n) = \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & n \\ n & n^2+1 \end{pmatrix}.$$

Upon comparing the elements in the resulting products above, we find that $n^2+1$ is the largest element, and that $n^2+n+1$ is the largest row sum, appearing in both $A(n)B(n)$ and $B(n)A(n)$. Hence, our base case holds.[8]

We now assume our proposition to be true for sequences of $m = 2k \geq 2$ alternating matrices, and we want to show it is true for sequences of $m+2$ alternating matrices. To begin with, we will look at how multiplication with $A(n)$ and $B(n)$ affects each row in a matrix. We may consider multiplication with $A(n)$ and $B(n)$ as corresponding to the following transformations:

1. transformation $R$ which takes the row vector $(x, y)$ to $(x, y+nx)$, which corresponds to multiplication on the right with $A(n)$,

2. transformation $L$ which takes row vector $(x, y)$ to $(x+ny, y)$, which corresponds to multiplication on the right with $B(n)$.

These two transformations give rise to two different recurrence relations, with $R$ resulting in the following relation:

$$X_m = X_{m-1}$$
$$Y_m = Y_{m-1} + n \cdot X_{m-1}$$

and $L$ the following:

[8]Note that we have chosen to look at the alternating sequence on the form $A(n)B(n)\ldots$. The theorem also holds for the sequence $B(n)A(n)\ldots$ and the proof is analogous, with the only difference that the largest row sum and element will be found in the lower row.

$$X_m = X_{m-1} + n \cdot Y_{m-1}$$
$$Y_m = Y_{m-1}.$$

Let $(X_m, Y_m)$ be the row vector with the largest element and sum in the matrix arising from $m$ alternating transformations on the form $RL$. From the recurrence relations we can easily see that this means that $X_m > Y_m$ [9]. We are now interested to find what next transformation gives the largest element and sum for our row vector. The $(m+1)$-th transformation must result in one of the following products:

$$\begin{pmatrix} X_{m+1} & Y_{m+1} \end{pmatrix} = \begin{pmatrix} X_m & Y_m \end{pmatrix} L = \begin{pmatrix} X_m & Y_m \end{pmatrix} \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} = \begin{pmatrix} X_m + n \cdot Y_m & Y_m \end{pmatrix}$$

$$\begin{pmatrix} X_{m+1} & Y_{m+1} \end{pmatrix} = \begin{pmatrix} X_m & Y_m \end{pmatrix} R = \begin{pmatrix} X_m & Y_m \end{pmatrix} \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} X_m & n \cdot X_m + Y_m \end{pmatrix}.$$

Since $X_m > Y_m$, we have that $Y_m + n \cdot X_m$ is the largest element of the above, and $Y_m + (n+1) \cdot X_m$ is the largest row sum by the same argument.

From the recurrence relations we can see that the case where $Y_m > X_m$ is completely analogous to the above, with the only difference that applying $L$ gives rise to the largest element and row sum. Since:

$$Y_{m+1} = Y_m + n \cdot X_m > X_m = X_{m+1},$$

we have that when applying $L$ to $(X_{m+1}, Y_{m+1})$ we receive $X_{m+1} + n \cdot Y_{m+1}$ as the largest element and $X_{m+1} + (n+1) \cdot Y_{m+1}$ as the largest row sum by the exact same argument as above. Hence, if we want our row to grow as fast as possible, the most advantageous choices for transformations $m+1$ and $m+2$ are $R$ and $L$ respectively. (Corresponding to multiplication with $A(n)B(n)$.)

Finally, we need to make sure that no other row can somehow result in a larger maximum element or larger sum than $(X_m, Y_m)$ when applying the $(m+1)$-th or $(m+2)$-th transformation to it. Hence, let's suppose we have some other row $(X'_m, Y'_m)$ that is as large as possible while still having a smaller sum and smaller largest element than $(X, Y)$. If we can show that this row still remains smaller than $(X, Y)$, the same will definitely hold for rows with smaller elements. From our induction hypothesis, we know that $X'_m < X_m$ and $X'_m + Y'_m < X_m + Y_m$,

[9]If we would have that $Y_m > X_m$ despite our last transformation being $L$, we would have the impossible situation (given that $X_m, Y_m \geq 0$ and $n \geq 2$): $Y_{m-1} = Y_m > X_m = X_{m-1} + n \cdot Y_{m-1}$.

but we will let $Y'_m > Y_m$ in order to make $(X'_m, Y'_m)$ as large as possible. We have the following possible transformations for $(X'_m, Y'_m)$:

$$\begin{pmatrix} X'_{m+1} & Y'_{m+1} \end{pmatrix} = \begin{pmatrix} X'_m & Y'_m \end{pmatrix} L = \begin{pmatrix} X'_m & Y'_m \end{pmatrix} \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} = \begin{pmatrix} X'_m + n \cdot Y'_m & Y'_m \end{pmatrix} \tag{8}$$

$$\begin{pmatrix} X'_{m+1} & Y'_{m+1} \end{pmatrix} = \begin{pmatrix} X'_m & Y'_m \end{pmatrix} R = \begin{pmatrix} X'_m & Y'_m \end{pmatrix} \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} X'_m & n \cdot X'_m + Y'_m \end{pmatrix}. \tag{9}$$

When comparing $(X_{m+1}, Y_{m+1})$ with (8) we have:

$$X'_m + n \cdot Y'_m = X'_m + Y'_m + (n-1) \cdot Y'_m < X_m + Y_m + (n-1) \cdot X_m = Y_m + n \cdot X_m$$

since by assumption $X_m > X'_m$ and $X_m + Y_m > X'_m + Y'_m$. Hence, the largest element can still be found in $(X_{m+1}, Y_{m+1})$. By the same argument, we have that:

$$X'_m + Y'_m + n \cdot Y'_m < X_m + Y_m + n \cdot X_m,$$

showing that the largest row sum is also found in $(X_{m+1}, Y_{m+1})$.

Upon comparing (9) with $(X_{m+1}, Y_{m+1})$ we find:

$$n \cdot X'_m + Y'_m = (n-1) \cdot X'_m + X'_m + Y'_m < X_m + Y_m + (n-1) \cdot X_m$$

since by assumption $X_m > X'_m$ and $X_m + Y_m > X'_m + Y'_m$. Again, by the same argument we find for the row sums that:

$$n \cdot X'_m + X'_m + Y'_m < n \cdot X_m + X_m + Y_m.$$

The argument is analogous when $Y > X$, with the difference that we need to check the row $(X'', Y'')$ where $Y > Y''$, $X + Y > X'' + Y''$ and $X'' > X$. This actually applies to our $(m+2)$-th transformation, since $X_{m+1} > Y_{m+1}$. From this observation, we can conclude that no other row vector can overtake the alternating sequence in the $(m+2)$- th transformation.

In the product $A(n)B(n)$ we have a row with the property that it has both the largest row sum and element, namely the row $(n^2+1, n)$. By the induction above, we have in fact shown that this row will grow the fastest and remain the largest row of the matrix when applying an alternating sequence of transformations to it. Any other row $(X', Y')$, appearing either in the same matrix as our fastest growing row or in some other matrix resulting from some product of $A(n)$ and $B(n)$, will hence remain smaller. This completes the proof.

$\square$

We will now calculate the minimum length of products on $A(2)$ and $B(2)$ for which an entry larger than $p$ can occur. Since the matrix entries of the alternating product $C_m$ grow faster than the elements of any other matrix product, $C_m$ will be the first word containing an element larger than $p$. This also means that it is the shortest word capable of producing a collision with another word. Hence, we only need to consider powers of $C(2) = A(2)B(2)$ in our below proposition, where we investigate how long words must be before one element in the matrix product may exceed $p$. The proposition is based on an example in [4] but presented in a more elaborate version here.

**Proposition 10.2.** *Let $C(2) = A(2)B(2) = \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix}$, and denote the $n$-th power of $C(2)$ by $(C(2))^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix}$. We claim that as long as $n < log_{3+\sqrt{8}} p$, all elements in $(C(2))^n$ are smaller than $p$.*

*Proof.* We claim that for the $(n+1)$-th power of $C(2)$ we get:

$$\begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix} \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5a_n + 2b_n & 2a_n + b_n \\ 2a_n + b_n & a_n \end{pmatrix}$$

which yields the following recurrence relations:

$$\begin{aligned} a_n &= 5a_{n-1} + 2b_{n-1} \\ b_n &= c_n = 2a_{n-1} + b_{n-1} \\ d_n &= a_{n-1}. \end{aligned}$$

We prove the above recurrence relation by induction on $n$. We examine the base case $n = 1$:

$$(C(2))^1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix}$$

which holds. We now suppose our hypothesis holds for all powers $n_0 \leq n$ of $C(2)$, and prove that it holds for the $(n+1)$-th power:

$$\begin{aligned}
(C(2))^{n+1} &= \begin{pmatrix} 5a_{n-1} + 2b_{n-1} & 2a_{n-1} + b_{n-1} \\ 2a_{n-1} + b_{n-1} & a_{n-1} \end{pmatrix} \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 5(5a_{n-1} + 2b_{n-1}) + 2(2a_{n-1} + b_{n-1}) & 2(5a_{n-1} + 2b_{n-1}) + 1(2a_{n-1} + b_{n-1}) \\ 5(2a_{n-1} + b_{n-1}) + 2(a_{n-1}) & 2(2a_{n-1} + b_{n-1}) + a_{n-1} \end{pmatrix} \\
&= \begin{pmatrix} 5a_n + 2b_n & 2a_n + b_n \\ 2a_n + b_n & a_n \end{pmatrix},
\end{aligned}$$

showing that our claim holds.

We now continue to solve the recurrence relations. Subtracting $2b_n$ from $a_n$, we receive:

$$a_n - 2b_n = 5a_{n-1} + 2b_{n-1} - 2(2a_{n-1} + b_{n-1}) = a_{n-1}$$
$$\iff$$
$$2b_n = a_n - a_{n-1}$$

and hence we get for $b_{n-1}$:

$$2b_{n-1} = a_{n-1} - a_{n-2}.$$

Plugging in the obtained values in the recurrence relations we get:

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

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

We get the following characteristic equation for $a_n$:

$$cr^n - 6cr^{n-1} + cr^{n-2} = 0,$$

which is reduced to

$$r^2 - 6r + 1 = 0$$

with solutions $r = 3 \pm \sqrt{8}$. We are looking for a solution on the form $a_n = C_1(3+\sqrt{8})^n + C_2(3-\sqrt{8})^n$. We use initial values from $(C(2))^0$ (the identity matrix) and $(C(2))^1$:

$$\begin{aligned} 1 &= a_0 = C_1 + C_2 \\ 5 &= a_1 = C_1(3+\sqrt{8}) + C_2(3-\sqrt{8}). \end{aligned}$$

Solving for $C_1$ and $C_2$ we receive $C_1 = \frac{1}{2} + \frac{1}{\sqrt{8}}$ and $C_2 = \frac{1}{2} - \frac{1}{\sqrt{8}}$. This gives the general solution

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

Similarly we get for $b_0$ and $b_1$:

$$\begin{aligned} 0 &= b_0 = D_1 + D_2 \\ 2 &= b_1 = D_1(3+\sqrt{8}) + D_2(3-\sqrt{8}) \end{aligned}$$

with solutions $D_1 = \frac{1}{\sqrt{8}}$ and $D_2 = -\frac{1}{\sqrt{8}}$, which gives us the general solution

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

From this we see that $a_n$ is the fastest growing element when taking powers of $C(2)$. This entry is smaller than $p$ as long as $n < log_{3+\sqrt{8}}p$. $\square$

As we have already stated, for a collision to occur at least one entry in at least one of the matrix products must be at least $p$. This means, using the proposition above, that as long as $n < log_{3+\sqrt{8}}p$, we know that $C(2)^n$ fails to produce collisions with other words. We have that $(C(2))^n$ has length $2n$ (since $C(2) = A(2)B(2)$), and hence no two positive words of length less than or equal to $m$ can be equal as long as $m < 2log_{3+\sqrt{8}}p = log_{\sqrt{3+\sqrt{8}}}p$. The fact that words up to this length are definitely safe from collisions, simultaneously gives us a lower bound for the girth of the graph, leading us to the below corollary from [4].

**Corollary 10.2.1.** *There are no collisions of hash values, i.e. no collisions of words $u, v$ on the form $u(A(2), B(2)) = v(A(2), B(2))$, as long as $u$ and $v$ are of length less than $\log_{\sqrt{3+\sqrt{8}}}p$. In particular we have that the girth of the Cayley graph of the monoid generated by $A(2)$ and $B(2)$ is at least $\log_{\sqrt{3+\sqrt{8}}}p$.*

The results from the above calculations can be used for finding a suitable order of $p$. For example if $p$ is of the order $2^{256}$, there will be no collisions of positive words $u$ and $v$ of length less than 203. As this is accounts for $\sum_{k=0}^{202} 2^k = 2^{203} - 1$ different hash values, upon comparing with for example SHA-1 which produces a 160-bit hash value (and thereby according the the pigeon hole principle has $2^{160}$ different possible hash values), we realize that this is a satisfying result.

# 11 Conclusion

We have introduced the concept of hash functions in general and in particular Zémor's Cayley hash function which uses matrices $A(1)$ and $B(1)$. Further, we have seen how this first hash function became subject to the lifting attack, which lifts the representation problem to $SL(2, \mathbb{Z}_+)$ before using the rather simple Euclid's algorithm for factoring. This attack did, however, prove to be futile in breaking the hash functions introduced by Bromberg. Although capable of producing attacks towards the group generated by $A(2)$ and $B(2)$, the lifting attack fails to find relations between words in the *monoid* generated by the same elements, and therefore the hash function remains safe. We also calculated a lower bound on the length of collisions for the hash function corresponding to $A(2)$ and $B(2)$, where we found that if $p$ is of the order $2^{256}$, no collisions of positive words of length less than 203 will occur. Finally, according to Bromberg, at this time there are no known attacks to the hash function corresponding to $A(2)$ and $B(2)$ nor to the one corresponding to $A(3)$ and $B(3)$, and hence the security of these hash functions is - to the best of our knowledge - intact [4].

## References

[1] L. Bromberg, *Cryptographic hash functions and some applications to information security* in Combinatorial and additive number theory II, 2017, 85-97.

[2] J.-P. Tillich and G. Zémor, *Group-theoretic hash functions* in Algebraic Coding, 1993, 90-110.

[3] J.-P. Tillich and G. Zémor, *Hashing with $SL_2$* in CRYPTO 1994, 1994, 40-49.

[4] L.Bromberg, V. Shrilrain and A. Vdovina, *Navigating in the Cayley graph of $SL_2(\mathbb{F}_p)$*, Semigroup Forum, 2017.

[5] C. Petit, *On graph-based cryptographic hash functions*, Université Catholique de Louvain, 2009.

[6] B. Sosnovski, *Cayley Graphs of Semigroups and Applications to Hashing*, City University of New York (CUNY), 2016.

[7] J. Hoffstein, J. Pipher, J. H. Silverman, *An Introduction to Mathematical Cryptography*, Springer, 2014.