Collisions for the LPS expander graph hash function

Jean-Pierre Tillich[1] and Gilles Zémor[2]

[1] INRIA, Équipe SECRET,
Rocquencourt, 78153 Le Chesnay, France.
jean-pierre.tillich@inria.fr
[2] Institut de Mathématiques de Bordeaux,
Université Bordeaux 1,
351, cours de la Libération, 33405 Talence, France
Gilles.Zemor@math.u-bordeaux1.fr

Abstract. We analyse the hash function family based on walks in LPS Ramanujan graphs recently introduced by Charles et al. We present an algorithm for finding collisions that runs in quasi-linear time in the length of the hashed value. A concrete instance of the hash function is considered, based on a 100-digit prime. A short collision is given, together with implementation details.

1 Introduction

Given the recent profusion of efficient attacks against widely used practical hash functions, among which the MD and SHA families, there is a growing need for hash functions built upon different principles, and in particular with some degree of proven collision resistance that would come under the form: finding a collision is equivalent to solving a clearly identified mathematical problem. A promising design strategy that has been experimented with in the past and is undergoing recent developments [2, 3], consists of choosing a large fixed graph that has a short and computationally efficient description, together with a natural correspondence between strings over a given alphabet and paths in the graph. The output of the hashed function is declared to be the endpoint of the path. Finding collisions is then essentially equivalent to finding cycles in the graph. If the hashed values have to be written with at least n bits, then the smallest cycle size (the girth of the graph) can be made to be at least cn for constant c, so that general purpose algorithms for finding cycles in graphs are useless because they are exponential in the cycle size.

Hash functions based on this principle were introduced in the past in [18, 17, 19]. The graphs are Cayley graphs over the groups G of 2×2 matrices SL_2 over finite fields of prime orders [18, 19] and order a power of 2 [17]. A Cayley graph over the group G has the group elements as vertex set and there is an edge between group elements x and y if $y = xs$ where s belongs to a small, fixed, carefully chosen set $\mathcal{S}$ of group elements.

Finding collisions in such schemes is tantamount to factoring group elements into short non-trivial products of elements of $\mathcal{S}$. The first attempt [18] was broken in [16]. Attacks have then been mounted against [19, 17] in [4, 6, 14]. However, a close look at these papers shows that they do not find genuine collisions. Geiselman [6] does discuss a method, but it produces collisions between input messages of astronomical size. Charnes and Pieprzyk [4] choose the field $\mathbb{F}_p$ that defines the hash function *after* choosing a potential collision. Similarly, Steinwandt et al. [14] choose the polynomial $P(X)$ over $\mathbb{F}_2$ that defines the hash function a posteriori. This means that if the defining parameters of the hash function (the prime p or the polynomial $P(X)$) are for example chosen to be the output of a trusted one-way function, no method is known to date for breaking these schemes. This is encouraging for SL_2 based hash functions and more generally, for hashing schemes whose collision resistance is based on the hardness of factoring in arithmetic groups.

Hashing schemes that build upon these ideas have also been proposed and discussed in [1, 15]. An application to authenticating sequences and signing video images is given in [12].

In recent work [2, 3], Charles et al. presented two hash function families that are also based on walking the message through a graph with arithmetic properties. The emphasis is on the expanding quality of the associated graph. Expansion is relevant to the hashing scheme because it implies the rapidly-mixing property, which means that when the input messages are sufficiently random, the output is uniformly distributed over the set of hash codes. This property stays true even when the input messages are limited to relatively small lengths. A proof of this property is clearly desirable for hashing, especially so if the hash functions are used in protocols whose security relies on the random oracle model.

In the present paper we consider the second hashing scheme of Charles et al., which is the fastest and the most likely to be considered for actual use [10]. Specifically, this scheme is based on the celebrated "Ramanujan" expander graphs of Lubotzky, Philips and Sarnak. In [3] the scheme is claimed to be an improvement over [19, 17] and the underlying theoretical problem believed to be difficult. In what follows we solve the underlying problem, namely the factorisation of unity into generators of the Ramanujan Cayley graph, and provide collisions for arbitrary instances of the LPS Ramanujan hashing scheme of [3]. We exhibit an algorithm for finding collisions that runs in time quasi-linear in n, where n is the hashcode size. An actual example is discussed and implementation details are given.

The paper is organised as follows. In Section 2 we give a precise description of the hash function of [3]. In Section 3 we give an overview of the attack together with the arithmetic properties that we need. In Section 4 we provide missing details. In Section 5 we discuss a worked-out example for a Ramanujan graph based on a 100-digit prime. Dealing with 1024-bit primes is not a problem but putting the factorisation in print would be ungainly and uninformative. In Section 6 we give some comments on the attack, on possible repairs, and more generally on hashing schemes based on factoring in groups. A Maple program implementing the attack is given in the appendix.

2 The hash function

The cryptographic function under study, that will be denoted by H, is a particular instance of the following construction.

2.1 The general construction

Defining parameter.

A finite group G, and a set of generators $\mathcal{S}$ such that $\mathcal{S}^{-1} = \mathcal{S}$. Let $a \stackrel{\text{def}}{=} |\mathcal{S}|-1$. Choose now a function:

$$\pi : \{0, 1, \ldots, a-1\} \times \mathcal{S} \to \mathcal{S}$$

such that for any $g \in \mathcal{S}$ the set $\pi(\{0, 1, \ldots, a-1\} \times \{g\})$ is equal to $\mathcal{S} \setminus \{g^{-1}\}$.
Algorithm.

Convert the input message to a base-a number $x_0 x_1 \ldots x_k$. Define the sequence $(g_i)_{0 \le i \le k}$ inductively as follows

$$g_i = \pi(x_i, g_{i-1})$$

where g_{-1} is some fixed element in $\mathcal{S}$. The hashcode of the input message is just the group element

$$H(\mathbf{x}) = g g_0 g_1 \ldots g_k$$

where g is a fixed element of G and $\mathbf{x} = (x_0, x_1, \ldots, x_k)$.

This construction is slightly more complex than the one presented in [17, 19]. The idea is roughly the same: replace the symbols of the text to be hashed with group elements and multiply them together to obtain the hashed value. What is different here is the fact that the way a group element is mapped to a letter in the text depends both on the letter and on the previous associated group element. This rather involved definition is a consequence of the fact that in the generator set $\mathcal{S}$ there are pairs of generators which are inverse of each other. This implies that in order to avoid trivial collisions one should avoid having products $g_i g_{i+1}$ where $g_i = g_{i+1}^{-1}$. The way π is defined on the set $\{0, 1, \ldots, a-1\} \times \{g\}$ avoids this unwanted phenomenon.

It can be checked [3] that finding collisions for the hash function reduces to the following problem

Problem 2.1 — *Find $g_1, g_2, \ldots, g_t$ all in $\mathcal{S}$ such that*

$$\begin{aligned} g_1 g_2 \ldots g_t &= \mathbf{1} \\ g_i g_{i+1} &\neq \mathbf{1} \ \ \forall i \in \{1, 2, \ldots, t-1\} \end{aligned}$$

More precisely, finding a collision for H with two messages of size t' and t'' gives a solution to the previous problem for some $t \leq t' + t''$. This can checked as follows. The hashed value of the first message is the result of a product of the form $gg'_1 \dots g'_{t'}$ whereas the second one corresponds to a product $gg''_1 \dots g''_{t''}$ where the g'_i's and the g''_i's belong to $\mathcal{S}$. Both values coincide and therefore $g'_1 \dots g'_{t'} = g''_1 \dots g''_{t''}$. This implies that $g'_1 \dots g'_{t'} g''^{-1}_{t''} \dots g''^{-1}_1 = \mathbf{1}$. Conversely, consider a factorisation of the form $g_1 g_2 \dots g_t = \mathbf{1}$ with $g_1, g_2, \dots, g_t$ all in $\mathcal{S}$ and $g_i g_{i+1} \neq \mathbf{1}$ for i in $\{1, 2, \dots, t-1\}$. This implies that $g_1 g_2 \dots g_{t'} = g_t^{-1} \dots g_{t'+1}^{-1}$ for any t' in $\{1, \dots t-1\}$. If g_1^{-1} and g_t are both different from g_{-1} this yields a collision for the hash function with messages of size t' and $t - t'$ respectively. Otherwise, if $|\mathcal{S}| \geq 4$ there exists g' in $\mathcal{S}$ such that $g' \in \mathcal{S} \setminus \{g_{-1}^{-1}, g_1^{-1}, g_t\}$. Observe now that $gg' g_1 g_2 \dots g_{t'} = gg' g_t^{-1} \dots g_{t'+1}^{-1}$ and that the first term corresponds to the hashed value of a message of size $t'+1$ whereas the second term corresponds to the hashed value of a message of size $t - t' + 1$.

As explained in the introduction, the *Cayley graph* associated to G and $\mathcal{S}$ has vertex set G and there is an edge between x and y if and only if y is equal to $x.g$ for some g in $\mathcal{S}$. Calculating the hashcode $gg_1 g_2 \dots g_t$ of a t-symbol long input message amounts to taking a non-backtracking walk in the graph by starting at vertex g and performing the following steps

$$g \stackrel{g_1}{\rightarrow} gg_1 \stackrel{g_2}{\rightarrow} gg_1 g_2 \rightarrow \cdots \stackrel{g_t}{\rightarrow} gg_1 g_2 \dots g_t.$$

This walk is non-backtracking since we do not allow products of the form $g_i g_{i+1}$ that would be equal to the identity.

The particular Cayley graph chosen by the authors of [3] (whose defining group G and generator set $\mathcal{S}$ are presented in Subsection 2.2 below) is the celebrated Ramanujan graph construction of Lubotzky, Phillips, Sarnak (LPS) [11]. It has two properties relevant to hashing.

First, the graph is a good expander (see [3], [13] for details and [9] for a modern survey on expander graphs). This implies among other things that a random walk in this graph is close to the uniform distribution when the length of the walk is some constant times the logarithm of the number of vertices. From this property it is readily seen that the distribution of the hashed values is close to the uniform distribution as soon as the text size is some constant times the hashcode size. In the Ramanujan graph case, the size of the constant is quite small (slightly above 2 will do the job here).

Second, the LPS graph has no small cycles. This ensures that solutions to Problem 2.1 are large enough to make exhaustive search hopeless.

2.2 The particular choice of [2, 3]

The authors of [2, 3] choose for G the group $\mathrm{PSL}_2(\mathbb{F}_p)$: recall that $\mathrm{SL}_2(\mathbb{F}_p)$ is the group of 2×2 matrices of determinant 1 with entries in $\mathbb{F}_p$ and $\mathrm{PSL}_2(\mathbb{F}_p)$ is obtained from $\mathrm{SL}_2(\mathbb{F}_p)$ by taking the quotient by its centre, that is $\{1, -1\}$. This amounts to identifying matrix A with $-A$. The group $\mathrm{PSL}_2(\mathbb{F}_p)$ is of size $p(p^2-1)/2$. The prime p is chosen congruent to 1 modulo 4. The size of the generator set $\mathcal{S}$ will be equal to $\ell+1$ where ℓ is a small prime congruent to 1 modulo 4 and which is also a quadratic residue $\pmod p$. The generators are obtained from the $\ell+1$ integer solutions (a,b,c,d) of the Diophantine equation

$$\begin{cases} a^2+b^2+c^2+d^2=\ell \\ \qquad a>0, \quad a\equiv 1 \pmod 2 \\ \qquad b\equiv c\equiv d\equiv 0 \pmod 2 \end{cases} \tag{1}$$

For a proof that the number of solutions to (1) is indeed $\ell+1$ see for example [5, Ch. 2]. To each such (a,b,c,d) we associate the 2×2 matrix with entries in the ring $\mathbb{Z}[i]$ of Gaussian integers

$$M(a,b,c,d)=\begin{pmatrix} a+ib & c+id \\ -c+id & a-ib \end{pmatrix}. \tag{2}$$

We then map the entries of $M(a,b,c,d)$ to elements of $\mathbb{F}_p$ by applying the ring homomorphism

$$\begin{aligned} \phi\ &: \mathbb{Z}[i]\to\mathbb{F}_p \\ a+ib &\mapsto a+\iota b \end{aligned} \tag{3}$$

where ι is a square root of -1 modulo p (which lies in $\mathbb{F}_p$ since $p\equiv 1 \pmod 4$). After applying ϕ we denote the resulting matrices by $\widetilde{M}(a,b,c,d)$. Note that

Fact 2.2 — *The determinant of $\widetilde{M}(a,b,c,d)$ is equal to $\ell \pmod p$.*

We now view the matrices $\widetilde{M}(a,b,c,d)$ as elements of $\mathrm{PGL}_2(\mathbb{F}_p)$. Recall that this is the group of 2×2 invertible matrices with entries in $\mathbb{F}_p$ obtained after identifying pairs of matrices M and N whenever there exists $\lambda\in\mathbb{F}_p$ such that $M=\lambda N \pmod p$.

The set of generators $\mathcal{S}$ is now declared to be the set of matrices $\widetilde{M}(a,b,c,d)$ in $\mathrm{PGL}_2(\mathbb{F}_p)$.

Note that $\mathcal{S}^{-1}=\mathcal{S}$. This comes from the fact that in $\mathrm{PGL}_2(\mathbb{F}_p)$ we have

$$\begin{aligned} \widetilde{M}(a,b,c,d)\widetilde{M}(a,-b,-c,-d) &= \begin{pmatrix} a+\iota b & c+\iota d \\ -c+\iota d & a-\iota b \end{pmatrix}\begin{pmatrix} a-\iota b & -c-\iota d \\ c-\iota d & a+\iota b \end{pmatrix} \\ &= \begin{pmatrix} a^2+b^2+c^2+d^2 & 0 \\ 0 & a^2+b^2+c^2+d^2 \end{pmatrix} \\ &\equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$

Finally, it should also be noted that $\mathcal{S}$ does not generate the whole group $\mathrm{PGL}_2(\mathbb{F}_p)$. This comes from the fact that all the matrices in $\mathcal{S}$ have determinant ℓ which is a square modulo p. Therefore only matrices with determinant that are quadratic residues $\pmod p$ are generated. It can be checked that the generated subgroup G is isomorphic to $\mathrm{PSL}_2(\mathbb{F}_p)$ (see [11, 13]).

The Cayley graph associated to G and $\mathcal{S}$ is denoted by $X_{\ell,p}$ and is the LPS Ramanujan graph mentioned above. Apart from its expansion properties, the graph $X_{\ell,p}$ has a girth (smallest cycle length) at least $2\log_\ell p$ (see [11, 13]). Practical sizes of the parameters would be a prime p of several hundred bits (say 1024) and a small prime ℓ (say 5). This would mean that the smallest solution to Problem 2.1 must involve at least 882 generators.

3 An outline of the attack

Factoring in $\mathrm{PGL}_2(\mathbb{F}_p)$ directly seems difficult. Our strategy will be to first lift matrices of $\mathrm{PGL}_2(\mathbb{F}_p)$ into a set of matrices with entries in $\mathbb{Z}[i]$, and then factor into a product of lifted generators of $\mathcal{S}$, namely the matrices of (2). The relevant set of matrices is

$$\Omega = \left\{ \begin{pmatrix} a+ib & c+id \\ -c+id & a-ib \end{pmatrix} \middle| \, (a,b,c,d) \in E_w \text{ for some integer } w>0 \right\}$$

where E_w is the set of 4-tuples $(a,b,c,d)\in\mathbb{Z}^4$ such that

$$\begin{cases} a^2+b^2+c^2+d^2 = \ell^w \\ a>0, \quad a\equiv 1 \pmod 2 \\ b\equiv c\equiv d\equiv 0 \pmod 2. \end{cases} \tag{4}$$

Consider now the set Σ of $\ell+1$ matrices $M(a,b,c,d)$ with $\mathbb{Z}[i]$ entries defined in (2). In other words, Σ is the subset of Ω corresponding to 4-tuples (a,b,c,d) in E_1, and it is also the lifted version of the set of generators $\mathcal{S}$. It turns out that the set Ω coincides, up to multiplication by ±1 and by powers of ℓ, with products of elements of Σ. Precisely, we have the following lemma which is a reformulation of Corollary 3.2 of [11]: for more details, see also [13, Lemma 2.5.4] or [5, Corollary 2.6.14].

Lemma 3.1 — *Any matrix M in Ω can be expressed in a unique way as a product*

$$M = \pm\ell^r M_1 M_2 \dots M_e$$

where $\log_\ell(\det M) = e+2r$ and the M_i's all belong to Σ and $M_iM_{i+1}\neq \ell\,\mathbf{1}$ for $i\in\{1,\dots,e-1\}$.

The attack now proceeds along the following lines.

Step 1 (lifting the identity in Ω): The aim of this step is to find a matrix M in Ω which is not of the form $\ell^r\mathbf{1}$ and such that if we replace the complex entries by their "corresponding" values in $\mathbb{F}_p$ (i.e. apply the mapping ϕ (3)) then we obtain a matrix $\widetilde{M}$ of the form $\lambda\mathbf{1}$. This amounts to finding a,b,c,d in $\mathbb{Z}$ such that

$$\begin{cases} a^2+b^2+c^2+d^2 = \ell^w \\ a>0, \quad a\equiv 1 \pmod 2 \\ b\equiv c\equiv d\equiv 0 \pmod{2p} \\ b^2+c^2+d^2\neq 0 \end{cases} \tag{5}$$

for some positive integer w.

Step 2 (factorisation step): Find the factorisation of M promised by Corollary 3.1:

$$M = \pm \ell^r M_1 M_2 \dots M_e.$$

What really makes this task simple is the fact that the factorisation is unique. Proceed as follows. We first find the greatest integer r such that ℓ^r divides the 4 entries of M. Let M' be such that $M = \ell^r M'$. We denote by $G_1, \dots, G_{\ell+1}$ the $\ell + 1$ elements forming the lifted generator set Σ.

We start by finding the rightmost element in the factorisation of M' by computing all products of the form $\pm M' G_i^{-1}$. Necessarily one of these products will be in Ω: it will correspond to the factorisation of M' where we have dropped the last element of the factorisation. It is impossible that there is more than one of these products which lies in Ω: by using the fact that every element of Ω can be factored into elements of Σ we would obtain at least two different factorisations for M'. This would contradict the unique factorisation property. Therefore, the unique G_i for which $\pm M' G_i^{-1}$ belongs to Ω is the last element of the factorisation of M'. Notice that checking whether a product $\pm M' G_i^{-1}$ is in Ω or not is computationally easy given the definition of Ω. We continue this process and it has to stop after $\log_\ell(\det M) - 2r$ steps because at each iteration the determinant of the left part of the factorisation gets divided by ℓ and because M' is of determinant ℓ^{w-2r}. The complexity of this step is obviously proportional to the length of the factorisation, i.e. at most w. We will see below that we can choose w to be approximately $2 \log_\ell p$, so that the complexity is not more than $O(\log p)$.

Step 3 (final step) The point is that the matrix $\widetilde{M}$ with entries in $\mathbb{F}_p$ reduces to the identity in $\mathrm{PGL}_2(\mathbb{F}_p)$ and can be factored in this group by using the $\ell + 1$ generators of $\mathcal{S}$ as follows:

$$\mathbf{1} \equiv \widetilde{M} \equiv \widetilde{M_1}\widetilde{M_2} \dots \widetilde{M_e}$$

where $\widetilde{M_i}$ is the application of the aforementioned homomorphism ϕ to each entry of M_i (meaning $\widetilde{M_i}$ belongs to $\mathcal{S}$). This solves Problem 2.1.

4 Solving step 1

Solving Equation (5) seems to be easier when w is even. So let us arbitrarily set $w = 2k$ and let us write $b = 2px, c = 2py, d = 2pz$. We are looking for integer solutions (a, x, y, z) to the equation

$$a^2 + 4p^2(x^2 + y^2 + z^2) = \ell^{2k}.$$

This implies that

$$(\ell^k - a)(\ell^k + a) = 4p^2(x^2 + y^2 + z^2)$$

Let us choose $a = \ell^k - 2mp^2$ for some integer m. In this case, $(\ell^k - a)(\ell^k + a) = 2mp^2(2\ell^k - 2mp^2)$. Thus x, y, z should satisfy the equation

$$x^2 + y^2 + z^2 = m(\ell^k - mp^2) \tag{6}$$

Let us now specify how m and k are chosen. a should be positive and k as small as possible in order to minimise the length of the factorisation of the identity obtained at the end. We choose k to be the smallest integer such that $\ell^k - 4p^2 > 0$. We may then either choose $m = 1$ or $m = 2$ in order to keep a positive. We claim now that for either $m = 1$ or $m = 2$ the number $m(\ell^k - mp^2)$ is a sum of three squares. Let us recall Legendre's theorem (see [7]) which asserts that all integers are sums of 3 squares with the exception of the integers of the form $4^s(8t + 7)$ where s and t are integers. Assume that $m = 1$ does not work. In other words, $\ell^k - p^2$ is not a sum of three squares. This means that there exists s and t which are non-negative integers such that $\ell^k - p^2 = 4^s(8t + 7)$. Note that s has to be positive in this case. Observe now that $2(\ell^k - 2p^2) = 4^s(16t + 14) - 2p^2$. This number is neither a multiple of 4 nor odd. Therefore it can not be of the form $4^u(8v + 7)$. This shows that $m = 2$ is suitable for our purpose.

It remains now to find x, y, z which satisfy Equation (6). One way of achieving this goal is to subtract from $m(\ell^k - mp^2)$ a random x^2 and to hope that the result N is a sum of 2 squares. In this case there is a simple and efficient algorithm relying on Euclid's algorithm for finding y and z explicitly such that $y^2 + z^2 = N$. Fermat's theorem (see [7]) on sums of two squares says that a number is expressible as a sum of two squares if and only if its prime factors congruent to 3 modulo 4 occur with an even exponent. Our approach is to try to find values of x for which N is of the form $2^s p'$ where p' is a prime congruent to 1 modulo 4. When $m = 1$ for instance, we choose even values of x and since $\ell^k - p^2 \equiv 0 \mod 4$ we check whether $(\ell^k - p^2 - x^2)/4$ is a prime congruent to 1 modulo 4. This happens roughly with probability of order $O(1/\ln(\ell^k - p^2))$.

It remains to explain how we find y and z such that

$$y^2 + z^2 = N. \tag{7}$$

This is classical and can be done by using continued fraction expansion. We give the details for the sake of self-sufficiency and to explain implementation details. Let us recall that the convergents $\frac{p_n}{q_n}$ of the continued fraction expansion of a real number x are obtained inductively from the formulas

$$\begin{aligned}(p_{-1}, q_{-1}) &= (0, 1)\\(p_0, q_0) &= (1, 0)\end{aligned}$$

and for all nonnegative values of n for which $q_n x - p_n \neq 0$

$$\begin{aligned}a_n &= \left[-\frac{q_{n-1}x - p_{n-1}}{q_n x - p_n}\right]\\p_{n+1} &= a_n p_n + p_{n-1}\\q_{n+1} &= a_n q_n + q_{n-1}\end{aligned}$$

where [ ] denotes the integer part.

The sequence $(q_n)_{n\geq 0}$ is strictly increasing and the $\frac{p_n}{q_n}$ are very good rational approximations of x. They satisfy:

Proposition 4.1 — *We have:*

$$\left| x - \frac{p_n}{q_n} \right| \leq \frac{1}{q_n q_{n+1}}.$$

From Fermat's theorem and the identity

$$(a^2+b^2)(c^2+d^2) = (ac-bd)^2 + (bc+ad)^2$$

we know that in order to find integer solutions of Equation (7) we just need to solve this kind of equation when N is a prime congruent to 1 modulo 4. In this case, -1 is a quadratic residue modulo N. This fact is used as follows

Proposition 4.2 — *Let N be a prime congruent to 1 modulo 4, R be a square root of -1 modulo N and $\xi \stackrel{def}{=} \frac{R}{N}$. Let $\frac{p_i}{q_i}$ be the convergents associated to the continued fraction expansion of ξ. Let n be the unique integer such that $q_n < \sqrt{N} < q_{n+1}$. We have*

$$q_n^2 + (q_n R - p_n N)^2 = N.$$

Proof. First of all it should be noticed that such an n exists: the sequence of the q_i's is increasing and is defined up to the term q_j such that $q_j = N$. It follows from Proposition 4.1 that $\left|\frac{R}{N} - \frac{p_n}{q_n}\right| < \frac{1}{q_n q_{n+1}}$. Hence $\left|q_n \frac{R}{N} - p_n\right| < \frac{1}{q_{n+1}} < \frac{1}{\sqrt{N}}$ (because $q_{n+1} > \sqrt{N}$). This implies that $|q_n R - p_n N| < \sqrt{N}$. We also have $q_n < \sqrt{N}$. Putting both inequalities together we obtain $q_n^2 + (q_n R - p_n N)^2 < 2N$. Let us notice now that

$$\begin{aligned} q_n^2 + (q_n R - p_n N)^2 &\equiv q_n^2 + q_n^2 R^2 \pmod N \\ &\equiv 0 \pmod N \end{aligned}$$

Therefore, we necessarily have that $q_n^2 + (q_n R - p_n N)^2 = N$.

The exact complexity of this step is unclear, this is due to the problem of estimating the time complexity for finding an N whose prime factors congruent to 3 modulo 4 all occur with an even exponent. We can upper bound this quantity by the complexity for finding an N of the form $2^s q$, where q is a prime congruent to 1 modulo 4 and heuristic arguments based on the density of primes congruent to 1 modulo 4 indicate that the number of x's which have to be tried in order to find a proper N will be of order $O(\log p)$. The complexity for performing the continued fraction expansion is also of order $O(\log p)$. Therefore the total complexity of this step should be extremely low and will be of order $O(\log p)$. This has been confirmed experimentally (see Section 5).

5 An example of an attack

In this example we take $p = 10^{100} + 949$ which is the first prime $p > 10^{100}$ such that $p = 1 \mod 4$. Computations were done with a Maple program given in the appendix.

Next, consider the hash function corresponding to the graph $X_{5,p}$, i.e. we set $\ell = 5$, the first possible case since we must have $\ell = 1 \bmod 4$. It turns out that 5 is a quadratic residue for this choice of p. The 6 generators of $X_{5,p}$ are given by the matrices with $\mathbb{Z}[i]$ entries,

$$G_1 = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \quad G_2 = \begin{pmatrix} 1+2i & 0 \\ 0 & 1-2i \end{pmatrix} \quad G_3 = \begin{pmatrix} 1 & 2i \\ 2i & 1 \end{pmatrix}$$
$$G_4 = \begin{pmatrix} 1 & -2i \\ -2i & 1 \end{pmatrix} \quad G_5 = \begin{pmatrix} 1-2i & 0 \\ 0 & 1+2i \end{pmatrix} \quad G_6 = \begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix}$$

or by their images $\widetilde{G_j}$ in $\mathrm{PGL}_2(\mathbb{F}_p)$.

Step 1. We first look for a, b, c, d satisfying (5). We choose k to be the first integer larger than $\log_5(2p^2)$ to make the right hand side of (6) positive. We obtain $k = 287$. We then compute $5^k - p^2$ which is of the form $4u$ with u odd. As was quite likely, we have $u \neq 7 \mod 8$, which means that u can be expressed as a sum of three squares. Furthermore, it turns out that we have $u = 1 \mod 4$, so we try subtracting from u squares of the form $4v^2$ and test $u - 4v^2$ for primality. When we meet a prime, it will necessarily be congruent to 1 modulo 4, so that we will be able to express it as a sum of two squares. The first v such that $N = u - 4v^2$ is prime is $v = 1431$.

We then proceed to express N as a sum of two squares. We first find a square root R of -1 modulo N. We arbitrarily choose the root whose representation in $\{1, \ldots, N-1\}$ is largest. We then expand R/N into a continued fraction and compute the largest n such that p_n/q_n is the nth convergent of the continued fraction expansion and $q_n < \sqrt{N}$. In this particular case we find $n = 192$. We then set

$$\begin{aligned} x &= 2q_n \\ y &= 2(p_n N - q_n R) \\ z &= 4 \times 1431 = 5724 \end{aligned}$$

and obtain $5^k - p^2 = 4u = x^2 + y^2 + z^2$. We then set

$$\begin{aligned} a &= \sqrt{5^{2k} - 16up^2} \text{ which is an integer} \\ b &= 2px \\ c &= 2py \\ d &= 2pz \end{aligned}$$

Step 2. We now factor in Ω the matrix

$$M = \begin{pmatrix} a+ib & c+id \\ -c+id & a-ib \end{pmatrix}$$

into a product of G_j's. We know that there is a unique way to do this, and that the length of the factorisation is $2k = 574$, i.e.

$$M = \pm M_1 \ldots M_{2k}$$

where $M_j = G_{\sigma(j)}$ with $\sigma(j) \in \{1, 2, \dots, 6\}$. To compute σ, we first multiply on the right M by all six matrices $G_1 \dots G_6$, and test whether all entries are multiples of 5. When this happens we have found $G_{\sigma(574)}$. We compute $M' = MG^{-1}_{\sigma(574)} = \frac{1}{5} MG_{7-\sigma(574)}$ and proceed recursively, testing $M'G_j$ at most six times to obtain $\sigma(573)$, and so on. After 574 iterations were are left either with the identity matrix $\mathbf{1}$ or with $-\mathbf{1}$. The first 24 values $\sigma(1), \sigma(2), \dots, \sigma(24)$ of σ are

$$2, 4, 2, 3, 3, 3, 3, 1, 1, 4, 1, 5, 5, 5, 5, 1, 5, 1, 1, 1, 4, 1, 4, 6,$$

and the remaining 550 values are given by the array in figure 1, each row giving the next 25 values. We have exhibited the factorisation of unity

$$\begin{array}{l}
6, 2, 1, 2, 3, 2, 2, 3, 1, 1, 1, 3, 1, 2, 2, 1, 2, 6, 6, 6, 3, 1, 5, 4, 1,\\
4, 5, 1, 1, 3, 2, 3, 6, 5, 5, 5, 3, 3, 5, 5, 6, 2, 4, 1, 1, 5, 3, 1, 5, 1,\\
2, 1, 2, 1, 5, 6, 4, 1, 4, 4, 4, 6, 5, 1, 5, 3, 1, 2, 2, 4, 1, 4, 5, 4, 1,\\
3, 6, 3, 3, 1, 4, 6, 3, 5, 5, 6, 4, 6, 3, 3, 1, 2, 3, 3, 2, 4, 5, 3, 5, 4,\\
5, 4, 2, 2, 2, 4, 6, 4, 1, 1, 4, 2, 3, 1, 4, 5, 4, 6, 5, 5, 3, 1, 4, 5, 6,\\
2, 1, 2, 6, 2, 1, 3, 3, 2, 6, 6, 5, 1, 5, 3, 1, 5, 1, 5, 1, 2, 6, 3, 3, 1,\\
1, 1, 4, 2, 1, 1, 3, 5, 6, 4, 6, 2, 6, 6, 3, 6, 2, 6, 6, 6, 2, 4, 1, 2, 6,\\
5, 3, 1, 4, 1, 2, 6, 4, 4, 2, 4, 4, 2, 1, 2, 4, 4, 1, 2, 2, 2, 2, 6, 3, 2,\\
1, 2, 4, 2, 6, 2, 2, 4, 4, 1, 1, 1, 1, 2, 6, 2, 4, 5, 3, 2, 4, 1, 1, 1, 4,\\
2, 2, 1, 1, 1, 3, 1, 5, 6, 2, 4, 5, 5, 1, 4, 1, 3, 2, 6, 6, 4, 6, 4, 6, 4,\\
6, 3, 1, 1, 2, 6, 3, 2, 6, 6, 6, 3, 1, 2, 4, 2, 3, 3, 3, 3, 1, 1, 4, 1, 5,\\
5, 5, 5, 1, 5, 1, 1, 1, 4, 1, 4, 6, 6, 2, 1, 2, 3, 2, 2, 3, 1, 1, 1, 3, 1,\\
2, 2, 1, 2, 6, 6, 6, 3, 1, 5, 4, 1, 4, 5, 1, 1, 3, 2, 3, 6, 5, 5, 5, 3, 3,\\
5, 5, 6, 2, 4, 1, 1, 5, 3, 1, 5, 1, 2, 1, 2, 1, 5, 6, 4, 1, 4, 4, 4, 6, 5,\\
1, 5, 3, 1, 2, 2, 4, 1, 4, 5, 4, 1, 3, 6, 3, 3, 1, 4, 6, 3, 5, 5, 6, 4, 6,\\
3, 3, 1, 2, 3, 3, 2, 4, 5, 3, 5, 4, 5, 4, 2, 2, 2, 4, 6, 4, 1, 1, 4, 2, 3,\\
1, 4, 5, 4, 6, 5, 5, 3, 1, 4, 5, 6, 2, 1, 2, 6, 2, 1, 3, 3, 2, 6, 6, 5, 1,\\
5, 3, 1, 5, 1, 5, 1, 2, 6, 3, 3, 1, 1, 1, 4, 2, 1, 1, 3, 5, 6, 4, 6, 2, 6,\\
6, 3, 6, 2, 6, 6, 6, 2, 4, 1, 2, 6, 5, 3, 1, 4, 1, 2, 6, 4, 4, 2, 4, 4, 2,\\
1, 2, 4, 4, 1, 2, 2, 2, 2, 6, 3, 2, 1, 2, 4, 2, 6, 2, 2, 4, 4, 1, 1, 1, 1,\\
2, 6, 2, 4, 5, 3, 2, 4, 1, 1, 1, 4, 2, 2, 1, 1, 1, 3, 1, 5, 6, 2, 4, 5, 5,\\
1, 4, 1, 3, 2, 6, 6, 4, 6, 4, 6, 4, 6, 3, 1, 1, 2, 6, 3, 2, 6, 6, 6, 3, 1.
\end{array}$$

Fig. 1. the remaining 550 values $\sigma(25), \dots, \sigma(574)$

$$\mathbf{1} = \widetilde{G}_{\sigma(1)} \widetilde{G}_{\sigma(2)} \dots \widetilde{G}_{\sigma(574)}$$

in $\mathrm{PGL}_2(\mathbb{F}_p)$. This can easily be checked with the program given in the appendix. Running time is counted in seconds rather than minutes, and stays that way if p is replaced by a 1024-bit prime.

6 Comments

The attack presented here is somewhat reminiscent of the "density attack" (to the terminology of [17]) that was used in [16] to break the hashing scheme first proposed in [18]. In that attack the group unit element is first lifted into a "dense" subset of $\mathrm{SL}_2(\mathbb{Z})$ and then a factorisation algorithm is applied in $\mathrm{SL}_2(\mathbb{Z})$.

Can the "Ramanujan" hash function family be fixed so as to make the present attack unfeasible ? Well, there are several natural solutions to address this problem. One idea is to change the set of generators from $\mathcal{S}$ to $\mathcal{S}^2$ (we square the elements of $\mathcal{S}$) for example. That way the present attack will succeed only if one manages to lift the identity element of G onto a matrix of Ω that has a very special (and rare) factorisation into elements of $\mathcal{S}$. A similar idea is to reduce the set $\mathcal{S}$ by throwing away some generators. In this case though, one must be careful to ensure that the modified set of generators generates the same subgroup of $\mathrm{PGL}_2(\mathbb{F}_p)$ as the original generator set. It is also unclear what will happen to the expansion properties when modifying the hash function in this way, and more study is required to come up with suitable choices.

The very property that makes the graphs $X_{\ell,p}$ Ramanujan gave us a tool for mounting an attack, so resorting to these highly structured Cayley graphs may not be the best idea if one is to base a hash function on factoring in arithmetic groups like SL_2 (or PSL_2 or PGL_2). However, for hashing purposes, lesser guaranteed expansion properties may be sufficient. A promising result in that direction is the recent paper of Helfgott [8] which shows that, for any generating set $\mathcal{S}$ of $G = \mathrm{SL}_2(\mathbb{F}_p)$, any element of G can be expressed as a product of elements of $\mathcal{S}$ of length not more than $O(\log^c p)$. This falls somewhat short of the rapidly-mixing property, but it does guarantee that if any such Cayley graph is used as the basis of a hashing scheme, then over a set of relatively small-length input messages, the corresponding set of hashed values ranges over the whole group G.

References

1. K. S. Abdukhalikov and C. Kim. On the security of the hashing scheme based on SL2. In S. Vaudenay, editor, *Fast Software Encryption '98*, volume 1372 of *LNCS*, pages 93–102. Springer, 1998.
2. D.X. Charles, E.Z. Goren, and K.E. Lauter. Cryptographic hash functions from expander graphs. In *Second NIST cryptographic hash workshop*, Santa Barbara, USA, August 2006.
3. D.X. Charles, E.Z. Goren, and K.E. Lauter. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, to appear in print, published online, Sept. 15, 2007. Available from Springer `http://www.springerlink.com/content/cv4r187833614475/`
4. C. Charnes and J. Pieprzyk. Attacking the SL2 hashing scheme. In *ASIACRYPT '94*, volume 917 of *LNCS*, pages 322–330. Springer-Verlag, 1995.
5. G. Davidoff, P. Sarnak and A. Valette. *Elementary Number Theory, Group Theory, and Ramanujan Graphs.* Cambridge University Press, 2003.

6. W. Geiselmann. A note on the hash function of Tillich and Zémor. In *Fast Software Encryption '96*, volume 1039 of *LNCS*, pages 51–52. Springer-Verlag, 1996.
7. E. Grosswald. *Representation of integers as sum of squares*. Springer Verlag, 1985.
8. H. A. Helfgott. Growth and generation in SL2(Z/pZ), 2005. to appear in Annals of Math., `http://arxiv.org/abs/math/0509024`.
9. S. Hoory, N. Linial and A. Wigderson. Expander graphs and their applications. *Bull. AMS*, 43: 439–561, 2006.
10. K. E. Lauter, D. X. Charles, and E. Z. Goren. Hash function constructions from expander graphs, May 2007. United States Patent 20070098150, `http://www.freepatentsonline.com/20070098150.html`.
11. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
12. J. Quisquater and M. Joye. Authentication of sequences with the SL2 hash function: Application to video sequences. *Journal of Computer Security*, 5(3):213–223, 1997.
13. P. Sarnak. *Some applications of modular forms*. Cambridge U. Press, Cambridge, 1990.
14. R. Steinwandt, M. Grassl, W. Geiselmann, and T. Beth. Weaknesses in the SL2 hashing scheme. In *CRYPTO '00*, volume 1880 of *LNCS*, pages 287–299. Springer-Verlag, 2000.
15. V. Shpilrain. Hashing with polynomials. In *ICISC*, volume 4296 of *LNCS*, pages 22–28. Springer-Verlag, 2006.
16. J-P. Tillich and G. Zémor. Group-theoretic hash functions. In *Algebraic Coding*, volume 781 of *LNCS*, pages 90–110, 1993.
17. J-P. Tillich and G. Zémor. Hashing with SL_2. In *CRYPTO '94*, volume 839 of *LNCS*, pages 40–49. Springer, 1994.
18. G. Zémor. Hash functions and graphs with large girth. In *EUROCRYPT '91*, volume 547 of *LNCS*. Springer Verlag, 1991.
19. G. Zémor. Hash functions and Cayley graphs. *Designs, Codes and Cryptography*, 4:381–394, 1994.

7 Appendix

Setting up the parameters of the hash function.

```
> p:= nextprime(10^100):
> p mod 4;
                              3
> p := nextprime(p):
> p mod 4;
                              1
> with(numtheory):
> legendre(5,p);
                              1
```

Step 1.

```
> evalf(log(2*p^2)/log(5));
                         286.5659883
> k:=287;
```

```
                            k := 287
> Z:=5^287-p^2:
> Z mod 4;
                               0
> Z mod 8;
                               4
> u:=Z/4:
> u mod 4;
                               1
```

Finding a square that substracted to u yields a prime $N = 1 \mod 4$.

```
> for v to 10000 do if isprime(u-4*v^2)=true then print(v): fi: od:
                              1431
                              1794
                              4434
                              5610
                              6555
                              6666
                              8484
                              9405
> N:= u-4*1431^2:
> N mod 4;
                               1
```

Expressing the prime N as a sum of two squares.

```
> R:= Roots(X^2+1) mod N:
> T:=R[1][1]:
> cf := cfrac(T/N):
> n:=0:
> while nthdenom(cf,n+1)<evalf(sqrt(N)) do n:=n+1:  od: print(n);
                              192
> Q:=nthdenom(cf,192): P := nthnumer(cf,192):
> x:=2*Q:
> y:=2*(Q*T-P*N):
> z:=4*1431;
                           z := 5724
```

Checking that we have found three squares that sum to $4u$.

```
> x^2+y^2+z^2 -4*u;
                               0
```

Defining the matrix M.

```
> 5^(2*k)-4*p^2*(x^2+y^2+z^2):
> a:= sqrt(%):
> b:=2*p*x:
> c:=2*p*y:
> d:=2*p*z:
> with(LinearAlgebra):
```

```
> M := Matrix(2,2):
> M[1,1]:=a+I*b:
> M[2,2]:=a-I*b:
> M[1,2]:=c+I*d:
> M[2,1]:=-c+I*d:
```

Checking the length of the factorisation of M that should be $2k$.

```
> Determinant(M):
> eval(log(%)/log(5));
                              574
```

Step 2.
Define the set of generators $G_1, \dots, G_6$.

```
> G[1] := Matrix(2,2):
> G[1][1,1]:=1: G[1][1,2]:=2: G[1][2,1]:=-2: G[1][2,2]:=1:
> G[1];
                             [ 1  2]
                             [     ]
                             [-2  1]
> G[6] := Matrix(2,2):
> G[6][1,1]:=1: G[6][2,2]:=1: G[6][1,2]:=-2: G[6][2,1]:=2:
> G[2] := Matrix(2,2):
> G[2][1,1]:=1+2*I: G[2][2,2]:=1-2*I: G[2][1,2]:=0: G[2][2,1]:=0:
> G[5]:=Matrix(2,2):
> G[5][1,1]:=1-2*I: G[5][2,2]:=1+2*I: G[5][1,2]:=0: G[5][2,1]:=0:
> G[3]:=Matrix(2,2):
> G[3][1,1]:=1: G[3][2,2]:=1: G[3][1,2]:=2*I: G[3][2,1]:=2*I:
> G[4]:=Matrix(2,2):
> G[4][1,1]:=1: G[4][2,2]:=1: G[4][1,2]:=-2*I: G[4][2,1]:=-2*I:
```

The procedure that factors M into a product of G_i's.

```
> fact := proc(m,MM)
> local L,H,i,j,X:
> L:=[]: H[1]:=MM:
> for i to m do
> for j to 6 do
> X := Multiply(H[i],G[j]):
> if (X[1,1] mod 5)=0 and (X[1,2] mod 5)=0 and (X[2,1] mod 5)=0
> and (X[2,2] mod 5)=0 then
> H[i+1]:=X/5: L := [7-j,op(L)]: fi: od:
> od:
> print(H[m+1]); return(L);
> end proc:
```

The actual factorisation for this particular example.

```
> F:=fact(574,M);
                             [-1  0]
                             [     ]
                             [0  -1]
```

```
F := [2, 4, 2, 3, 3, 3, 3, 1, 1, 4, 1, 5, 5, 5, 5, 1, 5, 1, 1, 1, 4, 1, 4, 6,
  6, 2, 1, 2, 3, 2, 2, 3, 1, 1, 1, 3, 1, 2, 2, 1, 2, 6, 6, 6, 3, 1, 5, 4, 1,
  4, 5, 1, 1, 3, 2, 3, 6, 5, 5, 5, 3, 3, 5, 5, 6, 2, 4, 1, 1, 5, 3, 1, 5, 1,
  2, 1, 2, 1, 5, 6, 4, 1, 4, 4, 4, 6, 5, 1, 5, 3, 1, 2, 2, 4, 1, 4, 5, 4, 1,
  3, 6, 3, 3, 1, 4, 6, 3, 5, 5, 6, 4, 6, 3, 3, 1, 2, 3, 3, 2, 4, 5, 3, 5, 4,
  5, 4, 2, 2, 2, 4, 6, 4, 1, 1, 4, 2, 3, 1, 4, 5, 4, 6, 5, 5, 3, 1, 4, 5, 6,
  2, 1, 2, 6, 2, 1, 3, 3, 2, 6, 6, 5, 1, 5, 3, 1, 5, 1, 5, 1, 2, 6, 3, 3, 1,
  1, 1, 4, 2, 1, 1, 3, 5, 6, 4, 6, 2, 6, 6, 3, 6, 2, 6, 6, 6, 2, 4, 1, 2, 6,
  5, 3, 1, 4, 1, 2, 6, 4, 4, 2, 4, 4, 2, 1, 2, 4, 4, 1, 2, 2, 2, 2, 6, 3, 2,
  1, 2, 4, 2, 6, 2, 2, 4, 4, 1, 1, 1, 1, 2, 6, 2, 4, 5, 3, 2, 4, 1, 1, 1, 4,
  2, 2, 1, 1, 1, 3, 1, 5, 6, 2, 4, 5, 5, 1, 4, 1, 3, 2, 6, 6, 4, 6, 4, 6, 4,
  6, 3, 1, 1, 2, 6, 3, 2, 6, 6, 6, 3, 1, 2, 4, 2, 3, 3, 3, 3, 1, 1, 4, 1, 5,
  5, 5, 5, 1, 5, 1, 1, 1, 4, 1, 4, 6, 6, 2, 1, 2, 3, 2, 2, 3, 1, 1, 1, 3, 1,
  2, 2, 1, 2, 6, 6, 6, 3, 1, 5, 4, 1, 4, 5, 1, 1, 3, 2, 3, 6, 5, 5, 5, 3, 3,
  5, 5, 6, 2, 4, 1, 1, 5, 3, 1, 5, 1, 2, 1, 2, 1, 5, 6, 4, 1, 4, 4, 4, 6, 5,
  1, 5, 3, 1, 2, 2, 4, 1, 4, 5, 4, 1, 3, 6, 3, 3, 1, 4, 6, 3, 5, 5, 6, 4, 6,
  3, 3, 1, 2, 3, 3, 2, 4, 5, 3, 5, 4, 5, 4, 2, 2, 2, 4, 6, 4, 1, 1, 4, 2, 3,
  1, 4, 5, 4, 6, 5, 5, 3, 1, 4, 5, 6, 2, 1, 2, 6, 2, 1, 3, 3, 2, 6, 6, 5, 1,
  5, 3, 1, 5, 1, 5, 1, 2, 6, 3, 3, 1, 1, 1, 4, 2, 1, 1, 3, 5, 6, 4, 6, 2, 6,
  6, 3, 6, 2, 6, 6, 6, 2, 4, 1, 2, 6, 5, 3, 1, 4, 1, 2, 6, 4, 4, 2, 4, 4, 2,
  1, 2, 4, 4, 1, 2, 2, 2, 2, 6, 3, 2, 1, 2, 4, 2, 6, 2, 2, 4, 4, 1, 1, 1, 1,
  2, 6, 2, 4, 5, 3, 2, 4, 1, 1, 1, 4, 2, 2, 1, 1, 1, 3, 1, 5, 6, 2, 4, 5, 5,
  1, 4, 1, 3, 2, 6, 6, 4, 6, 4, 6, 4, 6, 3, 1, 1, 2, 6, 3, 2, 6, 6, 6, 3, 1]
```

Verification: checking that M is indeed expressed in this way.

```
> Id:=Matrix(2,2):
> Id[1,1]:=1: Id[2,2]:=1:
>
> t:=Id:
> for i to 574 do t:=Multiply(t,G[F[i]]): od:
> M+t;
                              [0  0]
                              [    ]
                              [0  0]
```