Full Cryptanalysis of LPS and Morgenstern Hash Functions

Christophe Petit[1⋆], Kristin Lauter[2] and Jean-Jacques Quisquater[1⋆⋆]

[1]UCL Crypto Group⋆⋆⋆, [2]Microsoft Research.

e-mails: christophe.petit@uclouvain.be,klauter@microsoft.com,jjq@uclouvain.be

Abstract Collisions in the LPS cryptographic hash function of Charles, Goren and Lauter have been found by Zémor and Tillich [16], but it was not clear whether computing preimages was also easy for this hash function. We present a probabilistic polynomial time algorithm solving this problem. Subsequently, we study the Morgenstern hash, an interesting variant of LPS hash, and break this function as well. Our attacks build upon the ideas of Zémor and Tillich but are not straightforward extensions of it. Finally, we discuss fixes for the Morgenstern hash function and other applications of our results.

1 Introduction

Hash functions are widely used in cryptographic applications such as commitment schemes, digital signatures schemes, message authentication codes or password encryption. Typically, a hash function is required to be preimage and collision resistant and to have nearly uniform output distribution. Due to the importance of cryptographic hash functions, the SHA family was designed as a NIST standard [2]. However, recently discovered vulnerabilities in SHA-1 [14] prompted NIST to launch a competition for a New Cryptographic Hash Algorithm [1].

The NIST competition is stimulating research on hash functions in the cryptographic community and a lot of new schemes have been recently designed and put forward. Particularly appealing from a theoretical point of view, some of these schemes are *provably secure*, in the sense that their security relates to the hardness of some mathematical problem [8,4,3,12]. A good reduction to a simply formulated mathematical challenge facilitates the evaluation process and increases the confidence once the function has resisted first cryptanalytic attempts. However, it also gives the cryptanalyst a clue to break the scheme, and is especially problematic if the mathematical challenge turns out to be easy.

⋆ Research Fellow of the Belgian Fund for Scientific Research (F.R.S.-FNRS). Part of this work was done while he was visiting Crypto Group at Computer Science Department, UCSD.

⋆⋆ Part of this work was done while visiting MIT (CSAIL-Theory of Computation)

⋆⋆⋆ A member of BCRYPT and ECRYPT networks

The LPS hash function proposed by Charles, Goren and Lauter is one of these constructions [3]. It has a particularly elegant design, as the hash computation can be interpreted as a random walk in the optimal expander graphs of Lubotzky, Philips and Sarnak [7]. Finding collisions for this function is finding cycles in the graphs, which also amounts to finding a non-trivial factorization of the identity in terms of some particular elements of a projective group of matrices (a problem we will call the *decomposition problem*). Charles, Goren and Lauter proposed this problem as potentially hard. A major step in the breaking of the LPS hash function has recently been performed by Zémor and Tillich [16] who produced collisions by actually solving this problem.

One of the main contributions of this paper is an efficient algorithm that finds preimages for the LPS hash function. As Zémor and Tillich did, we actually solve the underlying problem which was presumed hard. Both for efficiency considerations and because of these new attacks, it also seemed worthwhile to study the Morgenstern hash function, an interesting variant of the LPS hash relying on different graphs. A second main contribution of this paper is to adapt the Zémor and Tillich attack to Morgenstern hashes, and as an example we give an efficient collision finding algorithm. Combining the ideas of our two algorithms also gives a preimage finding algorithm for Morgenstern hashes, that we do not present here due to space limitations.

The paper is organized as follows: in Section 2 we describe the LPS and Morgenstern hash functions; in Section 3 we recall the Zémor and Tillich algorithm; Section 4 presents our preimage algorithm for LPS hashes; in Section 5 we adapt Zémor and Tillich's algorithm to Morgenstern hashes and in Section 6 we discuss fixes for LPS and Morgenstern hashes, as well as potential applications of our results. In the appendix we give toy and 1024−bit examples of our algorithms.

2 LPS and Morgenstern hash functions

A *Cayley graph* $\mathcal{C}_{G,S} = (V, E)$ is a graph constructed from a group G and a subset S of G as follows: V contains a vertex v_g associated to each element $g \in G$, and E contains the directed edge (v_{g_1}, v_{g_2}) iff there is some $s \in S$ such that $g_2 = g_1 s$. The elements of S are called the *graph generators*. The graph $\mathcal{C}_{G,S}$ is $|S|$-regular; it is connected iff S generates G; it is undirected iff $S = S^{-1}$.

A general construction for a cryptographic hash function from a Cayley graph was introduced by Zémor and Tillich [15,12,13] in the directed case and by Charles, Goren and Lauter [3] in the undirected case. In this paper, we focus on two instances of the undirected graph construction, which we now recall following mainly the description given in [16].

Let $a := |S| - 1$. We will define a function π which orders the set of generators (minus one generator to avoid back-tracking). Fix a function $\pi : \{0, 1...a-1\} \times S \to S$ such that for any $g \in S$ the set $\pi(\{0, 1...a-1\} \times \{g\})$ is equal to $S \setminus \{g^{-1}\}$. Let g_0 and g_{IV} be arbitrary fixed elements of S and G respectively. The input message is converted to a base a number $x_1...x_k$ and the elements $g_i = \pi(x_i, g_{i-1})$ are computed

recursively. The hashcode of the input message is the product of group elements $H(x) = g_{IV} g_1 ... g_k$.

We will call hash functions constructed following this design strategy *Cayley hashes*. These hash functions have some very interesting properties:

- The girth of the Cayley graph is the length of the smallest cycle, and no two distinct messages of the same length can collide if their length is less than half the girth.
- If the chosen graphs are good expanders (see [6] for precise definitions and applications), the outputs tend to be uniformly distributed, and the convergence to the uniform distribution is fast.
- Differential cryptanalysis (DC), which has been the most successful approach against SHA-1, does not seem to apply to Cayley hashes. Indeed, DC typically activates various portions of the message simultaneously, while in Cayley hashes the bits (or k-its) are processed one at the time.
- Collision resistance is equivalent to the hardness of a simply-stated representation problem in the corresponding group: namely, this problem is to find a factorization of the identity $1 = g_1 g_2 ... g_t$ with $g_i \in S$ and $g_i g_{i+1} \neq 1$ for all $i \in \{1, 2, ...t-1\}$.
- Preimage resistance and second preimage resistance follow from similar problems.

One proposal by Charles, Goren and Lauter [3] is to use the celebrated LPS graphs of Lubotzky, Philips and Sarnak [7] that we now describe. Let p and l be primes, l small and p large, both p and l equal to 1 mod 4, and l being a quadratic residue modulo p. To l and p is associated an LPS graph $X_{l,p}$ as follows. Let $\mathbf{i}$ be an integer such that $\mathbf{i}^2 \equiv -1 \bmod p$. The vertices of $X_{l,p}$ are elements in the group $G = PSL(2, \mathbb{F}_p)$ (*i.e.* 2×2 matrices of determinant, modulo the equivalence relation $M_1 \sim \lambda M_2, \lambda \in \mathbb{F}_p^*$). The set S is $S = \{g_j\}_{j=1...,l+1}$, where

$$g_j = \begin{pmatrix} \alpha_j + \mathbf{i}\beta_j & \gamma_j + \mathbf{i}\delta_j \\ -\gamma_j + \mathbf{i}\delta_j & \alpha_j - \mathbf{i}\beta_j \end{pmatrix}, \qquad j = 1, ..., l+1;$$

and $(\alpha_j, \beta_j, \gamma_j, \delta_j)$ are all the integer solutions of $\alpha^2 + \beta^2 + \gamma^2 + \delta^2 = l$, with $\alpha > 0$ and β, γ, δ even. The Cayley graph $X_{l,p} = \mathcal{C}_{G,S}$ is undirected since S is stable under inversion.

The choice of LPS graphs was very appealing : they are Ramanujan (*i.e.*, they have optimal expansion properties asymptotically [7,6]); they have no short cycles, and computing the resulting hash functions turned out to be quite efficient compared to other provable hashes [10]. Unfortunately, it turns out that LPS hash function is neither collision nor preimage resistant (see Sections 3 and 4 below).

For efficiency reasons, we recently considered the use of Morgenstern graphs to replace LPS graphs in Charles-Goren-Lauter's construction [10]. Morgenstern's Ramanujan graphs [9] generalize LPS graphs from an odd prime $p \equiv 1 \bmod 4$ to any

power of any prime q. More specifically, we suggested the use of Morgenstern graphs with $q = 2^k$, that we now describe.

Let q be a power of 2 and $f(x) = x^2 + x + \epsilon$ irreducible in $\mathbb{F}_q[x]$. Let $p(x) \in \mathbb{F}_q[x]$ be irreducible of even degree $n = 2d$ and let $\mathbb{F}_{q^n}$ be represented by $\mathbb{F}_q[x]/(p(x))$. The vertices of the Morgenstern graph Γ_q are elements of $G = PSL_2(\mathbb{F}_{q^n})$ (*i.e.* 2×2 matrices modulo the equivalence relation $M_1 \sim \lambda M_2, \lambda \in \mathbb{F}_{q^n}^*$). Let $\mathbf{i} \in \mathbb{F}_{q^n}$ be a root of $f(x)$. The set S is taken to be $S = \{g_j\}_{j=1...,q+1}$, where

$$g_j = \begin{pmatrix} 1 & \gamma_j + \delta_j \mathbf{i} \\ (\gamma_j + \delta_j \mathbf{i} + \delta_j)x & 1 \end{pmatrix}, \qquad j = 1, ..., q+1;$$

where $\gamma_j, \delta_j \in \mathbb{F}_q$ are all the $q+1$ solutions in $\mathbb{F}_q$ for $\gamma_j^2 + \gamma_j \delta_j + \delta_j^2 \epsilon = 1$. The Cayley graphs $\Gamma_q = \mathcal{C}_{G,S}$ are also undirected as each g_j has order 2.

An interesting property of Morgenstern hashes compared to LPS hashes is that arithmetic is done in fields that are extensions of $\mathbb{F}_2$ rather than in finite prime fields, potentially leading to faster hashes for some architectures. The total break of LPS hashes leads to the question of whether similar attacks can be found for Morgenstern hashes. This is indeed the case, and as an example we give a collision-finding attack for $q = 2$ in Section 5.

3 Zémor and Tillich algorithm

As our new attacks will build upon it, we now briefly recall Zémor and Tillich's algorithm that finds collisions for LPS hashes [16]. The algorithm lifts the graph generators and the representation problem from $PSL(2, \mathbb{F}_p)$ to an appropriate subset Ω of $SL(2, \mathbb{Z}[i])$ (in this section and the next one, i is the complex imaginary number satisfying $i^2 + 1 = 0$ while $\mathbf{i}$ is a solution to $\mathbf{i}^2 + 1 \equiv 0 \bmod p$). The relevant set is

$$\Omega = \left\{ \begin{pmatrix} a + bi & c + di \\ -c + di & a - bi \end{pmatrix} | (a, b, c, d) \in E_e \text{ for some integer } e > 0 \right\}$$

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

$$\begin{cases} a^2 + b^2 + c^2 + d^2 = l^e \\ a > 0, a \equiv 1 \bmod 2 \\ b \equiv c \equiv d \equiv 0 \bmod 2. \end{cases}$$

We will call the first of these equations describing E_e the *norm equation*, as the left-hand side of this equation is the norm of the quaternion corresponding to the quadruplet (a, b, c, d) (see [7]). The set Ω has two important properties: first, any element of Ω admits a unique factorization in terms of the lifts of the graph generators, and second, there exists a multiplicative homomorphism from Ω to $PSL(2, \mathbb{F}_p)$ that allows translation of this factorization back to $PSL(2, \mathbb{F}_p)$.

In their exposition, Zémor and Tillich decompose their attack into three steps. The first step (lifting the decomposition problem to $SL(2, \mathbb{Z}[i])$) amounts to finding

integers a, b, c, d and λ satisfying the following conditions:

$$\begin{cases} (a,b,c,d) \in E_e \\ (a,b,c,d) \quad \text{not divisible by l} \\ (a,b,c,d) \equiv \lambda(1,0,0,0) \bmod p. \end{cases}$$

Putting every congruence condition into the norm equation leads to a diophantine equation that was solved by Zémor and Tillich in their paper. The second step of the attack is to factorize the lifted element I' of Ω into products of lifted generators $g'_j, j = 1...l+1$. We know this factorization is unique and has size e, so let's write it $I' = g'_{j_1} g'_{j_2} ... g'_{j_e}$. Multiplying on the right by a lifted generator g' gives a matrix that is divisible by l if and only if $g' = (g'_{j_e})^{-1}$, so by trying each of the graph generators we get the last factor, and we then proceed recursively. The final step is to transpose the factorization of I' in Ω into a factorization of the identity in $PSL(2, \mathbb{F}_p)$, but using the homomorphism from Ω to $PSL(2, \mathbb{F}_p)$, this last step is trivial. For details on the attack we refer to [16].

4 Finding preimages for LPS hashes

Suppose we are given a matrix $M = \begin{pmatrix} M_1 & M_2 \\ M_3 & M_4 \end{pmatrix} \in PSL(2, \mathbb{F}_p)$ which has square determinant, and we are asked to find a preimage, that is a factorization of it with the graph generators. By solving two linear equations in $\mathbb{F}_p$ we can write it in the form

$$M = \begin{pmatrix} A + B\mathbf{i} & C + D\mathbf{i} \\ -C + D\mathbf{i} & A - B\mathbf{i} \end{pmatrix}.$$

Our algorithm follows along the lines of Zémor and Tillich's. We first lift the problem from $PSL(2, \mathbb{F}_p)$ to the set Ω defined above, then factorize in Ω and finally come back to $PSL(2, \mathbb{F}_p)$. The only difference will be in the first step. Lifting the representation problem now amounts to finding integers a, b, c, d and λ satisfying the following conditions:

$$\begin{cases} (a,b,c,d) \in E_e \\ (a,b,c,d) \quad \text{not divisible by l} \\ (a,b,c,d) \equiv \lambda(A,B,C,D) \bmod p. \end{cases}$$

We write $a = A\lambda + wp$, $b = B\lambda + xp$, $c = C\lambda + yp$ and $d = D\lambda + zp$ with $w, x, y, z \in \mathbb{Z}$. For convenience we choose e even, that is $e = 2k$ for k an integer. The norm equation becomes

$$(A\lambda + wp)^2 + (B\lambda + xp)^2 + (C\lambda + yp)^2 + (D\lambda + zp)^2 = l^{2k}. \tag{1}$$

In the case $B = C = D = 0$ the norm equation is $(A\lambda + wp)^2 + (xp)^2 + (yp)^2 + (zp)^2 = l^{2k}$ and was solved by Zémor and Tillich as follows: choose

$$A\lambda + wp = l^k + mp^2$$

for small m and appropriate k, hence the equation is already satisfied modulo p^2. Simplifying by p^2 we get a quadratic diophantine equation of type $x^2 + y^2 + z^2 = m(l^k - mp^2)$ which Zémor and Tillich show has a solution either for $m = 1$ or for $m = 2$. In Equation 1, when B, C, D are non-zero we cannot divide by p^2 because of the term $2p(wA + xB + yC + zD)\lambda$. Since we do not, the coefficients of degree-2 terms are huge (at least p), and the equation is at first sight very hard to solve.

We overcome this difficulty with a new idea. In the remainder of this section, we will solve the preimage problem for diagonal matrices with A and/or B non-zero, and then we will write any matrix as a product of four diagonal matrices and up to four graph generators. Altogether this leads to an efficient probabilistic algorithm that finds preimages of the LPS hash function.

Preimages for diagonal matrices Now we show how to find a factorization of a matrix

$$M = \begin{pmatrix} A + B\mathbf{i} & \\ & A - B\mathbf{i} \end{pmatrix}$$

such that $A^2 + B^2$ is a square modulo p. Write $y = 2y'$ and $z = 2z'$ where y', z' are integers. We need to find integer solutions to

$$\begin{cases} (A\lambda + wp)^2 + (B\lambda + xp)^2 + 4p^2(y'^2 + z'^2) = l^{2k} \\ A\lambda + wp \equiv 1 \bmod 2 \\ B\lambda + xp \equiv 0 \bmod 2 \end{cases}$$

Fix $k = \lceil \log_l(8p^2) \rceil$. As $A^2 + B^2$ is a square, there are exactly two values for λ in $\{0, 1, ...p - 1\}$ satisfying the norm equation modulo p:

$$(A^2 + B^2)\lambda^2 = l^{2k} \bmod p.$$

Choose either of them, and let $m := (l^{2k} - (A^2 + B^2)\lambda^2)/p$. Our strategy will be to pick random solutions to

$$\begin{cases} l^{2k} - (A\lambda + wp)^2 - (B\lambda + xp)^2 \equiv 0 \bmod p^2 \\ A\lambda + wp \equiv 1 \bmod 2 \\ B\lambda + xp \equiv 0 \bmod 2 \end{cases}$$

until the equation

$$4(y'^2 + z'^2) = n$$

has solutions, where

$$n := \left(l^{2k} - (A\lambda + wp)^2 - (B\lambda + xp)^2\right)/p^2.$$

A random solution to the congruence system is computed as follows: until you get x with the correct parity, pick a random $w \in \{0, 1, ...p-1\}$ with the right parity and compute $x = \frac{m}{2\lambda B} - \frac{A}{B}w \bmod p$. By the way k, x and w are chosen we are guaranteed that $n > 0$ so the equation $4(y'^2 + z'^2) = n$ has solution if and only

if 4 divides n and all prime factors of n congruent to 3 modulo 4 appear an even number of times in the factorization of n. To avoid the factorization of n in the algorithm, we will actually strengthen this condition to n being equal to 4 times a prime congruent to 1 modulo 4. When it has solutions, the equation $4(y'^2+z'^2)=n$ is easily solved with the Euclidean algorithm, as recalled in [16]. After lifting the problem to $SL(2,\mathbb{Z}[i])$ the second and third steps of the algorithm are the same as in Zémor-Tillich algorithm. So we are done with the factorization of diagonal matrices.

Reduction to the diagonal case Now we show how to decompose any matrix $M \in PSL(2,\mathbb{F}_p)$ into a product of diagonal matrices and graph generators. We may additionally assume that all the entries of M are nonzero: if they are not, just multiply M by gg^{-1} for some adequate g in S, and consider the factorization of $g^{-1}M$. We will show how to find (λ, α, ω, β_1, β_2) with the last four being squares, such that

$$\begin{pmatrix} M_1 & M_2 \\ M_3 & M_4 \end{pmatrix} = \lambda \begin{pmatrix} 1 & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} = \lambda \begin{pmatrix} f_1 & \omega f_2 \\ \alpha f_3 & \alpha\omega f_4 \end{pmatrix} \tag{2}$$

and

$$\begin{aligned} \begin{pmatrix} f_1 & f_2 \\ f_3 & f_4 \end{pmatrix} &= \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \beta_2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1-4\beta_1-4\beta_2-4\beta_1\beta_2 & 2-8\beta_1+2\beta_2+2\beta_1\beta_2 \\ -2-2\beta_1+8\beta_2-2\beta_1\beta_2 & -4-4\beta_1-4\beta_2+\beta_1\beta_2 \end{pmatrix}. \end{aligned}$$

Lemma 1. *Matrix equation (2) is equivalent to the following system:*

$$\begin{cases} M_2M_3f_1f_4 - M_1M_4f_2f_3 = 0 \\ \alpha M_1f_3 - M_3f_1 = 0 \\ \omega M_3f_4 - M_4f_3 = 0 \\ \lambda f_1 - M_1 = 0 \end{cases} \tag{3}$$

Proof : ($\Rightarrow$) Fourth equation is entry (1,1) of the matrix equation. Third equation is entry (2,1) times M_1 minus entry (1,1) times M_3. Second equation is entry (1,2) times M_1 minus entry (1,1) times M_2. First equation is entry (1,1) times entry (2,2) times M_2M_3 minus entry (1,2) times entry (2,1) times M_1M_4.
($\Leftarrow$) Last equation is $M_1 = \lambda f_1$ that is entry (1,1). We have $M_2 = \frac{M_1M_4f_2f_3}{M_3f_1f_4}$ by first equation so $M_2 = f_2\frac{M_4f_3}{M_3f_4}\frac{M_1}{f_1} = f_2\omega\lambda$ by third and fourth equation, that is entry (1,2). We have $M_3 = \frac{\alpha M_1f_3}{f_1} = \alpha\lambda f_3$ by second then fourth equation, that is entry (2,1). We have $M_4 = \omega M_3\frac{f_4}{f_3}$ by third equation, so using the already proved entry (2,1) we have $M_4 = \omega\alpha\lambda f_3\frac{f_4}{f_3} = \omega f_4\alpha\lambda$ that is entry (2,2). □

In the system of equations (3), the first equation only involves β_1 and β_2 while the other equations are linear once β_1 and β_2 are fixed. So we can concentrate on solving the first equation, which is quadratic in both β_1 and β_2:

$$\begin{aligned} M_2M_3f_1f_4 - M_1M_4f_2f_3 = {} & 4(M_2M_3 - M_1M_4)(-\beta_1^2+3\beta_1+4)\beta_2^2 \\ & + \left(M_2M_3(12\beta_1^2+49\beta_1+12) + M_1M_4(-12\beta_1^2+76\beta 1-12)\right)\beta_2 \\ & + 4(M_2M_3 - M_1M_4)\left(4\beta_1^2+3\beta_1-1\right). \end{aligned}$$

Our algorithm then proceeds as follows:

1. Pick a random β_1 which is a square.
2. Compute the discriminant of the quadratic equation in β_2, β_1. If it is not a square, go back to 1.
3. Solve the quadratic equation. If none of the roots is a square, go back to 1. Else, assign a quadratic root to β_2.
4. Compute f_1, f_2, f_3, f_4.
5. Solve $\alpha M_1 f_3 - M_3 f_1 = 0$ to get α. If α is not a square, go back to 1.
6. Solve $\omega M_3 f_4 - M_4 f_3 = 0$ to get ω. If ω is not a square, go back to 1.

This concludes the exposition of our algorithm.

Runtime analysis First consider the algorithm for diagonal matrices. Assuming n behaves "as a random number" then according to the prime number theorem we will need $\mathcal{O}(\log n) = \mathcal{O}(\log p)$ trials before getting one n of the correct form. For each trial, the most expensive computation is a primality test, which can be done in polynomial time (in our implementation, we actually use the probabilistic function *mpz_probab_prime_p* of GNU MP). So the algorithm for diagonal matrices is probabilistic polynomial time. In the reduction algorithm, the probability for a random number to be a square modulo p being one half, we estimate that a solution (λ, α, ω, β_1, β_2) with the last four being squares can be found in about 2^4 trials. Consequently, the whole algorithm is probabilistic polynomial time. Our implementation using GNU MP finds preimages in less than 2 minutes for 1024-bit parameters on an Intel Pentium M processor 1.73GHz.

5 Collisions for the Morgenstern hash function

Now we show how to adapt Zémor and Tillich's algorithm for finding collisions in Morgenstern hashes when $q = 2$. Our algorithm lifts the representation problem from $SL(2, \mathbb{F}_{2^n})$ to a subset Ω of $SL(2, \mathbb{A})$ where $\mathbb{A} = \mathbb{F}_2[x, y]/(y^2 + y + 1)$ (in this section, i will denote a root of $i^2 + i + 1 = 0$ in $\mathbb{A}$ while $\mathbf{i}$ is a root of the same equation in $\mathbb{F}_{2^n}$). The relevant set is

$$\Omega = \left\{ \begin{pmatrix} a + bi & c + di \\ x(c + di + d) & a + bi + b \end{pmatrix} | (a, b, c, d) \in E_e \text{ for some integer } e > 0 \right\}$$

where E_e is the set of 4-tuples $(a, b, c, d) \in \mathbb{F}_2[x]$ such that

$$\begin{cases} (a^2 + b^2 + ab) + (c^2 + d^2 + cd)x = (1 + x)^e \\ a \equiv 1 \bmod x \\ b \equiv 0 \bmod x. \end{cases}$$

Again call the first of these equations the *norm equation*. We point out that in this section, small letters a, b, c, d, i, p are polynomials in x over $\mathbb{F}_2$, while capitalized letters will be used for elements of the field $\mathbb{F}_{2^n}$. By [9], corollary 5.4 and 5.7, if we

restrict E_e to tuples (a,b,c,d) not divisible by $(1+x)$, the elements of Ω have a unique factorization in terms of the lifts of the graph generators:

$$g_0' = \begin{pmatrix} 1 & 1+i \\ ix & 1 \end{pmatrix}, \quad g_1' = \begin{pmatrix} 1 & 1 \\ x & 1 \end{pmatrix}, \quad g_2' = \begin{pmatrix} 1 & i \\ (1+i)x & 1 \end{pmatrix}.$$

Moreover, the "reduction modulo p" $(a,b,c,d) \to (A,B,C,D) = (a,b,c,d) \bmod p$ gives a homomorphism from Ω to $SL(2,\mathbb{F}_{2^n})$:

$$\begin{pmatrix} a+bi & c+di \\ x(c+di+d) & a+bi+b \end{pmatrix} \to \begin{pmatrix} A+B\mathbf{i} & C+D\mathbf{i} \\ x(C+D\mathbf{i}+D) & A+B\mathbf{i}+B \end{pmatrix}.$$

From this it is now clear how the second and third steps of Zémor and Tillich algorithm will work for Morgenstern hashes, so we now give details for first step. This amounts to lifting the representation problem, that is finding $a,b,c,d,\lambda \in \mathbb{F}_{2^n}$ satisfying the following conditions:

$$\begin{cases} (a,b,c,d) \in E_e \\ (a,b,c,d) \quad \text{not divisible by x+1} \\ (a,b,c,d) \equiv \lambda(1,0,0,0) \bmod p. \end{cases}$$

Write $b = xpb'$, $c = pc'$, $d = pd'$ for $b', c', d' \in \mathbb{F}_2[x]$ and arbitrarily choose $e = 2k$ and $a = (1+x)^k + xpm$, with $k \in \mathbb{Z}$ and $m \in \mathbb{F}_2[x]$ still to be determined. Note that such an a satisfies $a \equiv 1 \bmod x$. The norm equation becomes

$$x^2p^2m^2 + x^2p^2b'^2 + xpb'\left((1+x)^k + mxp\right) + xp^2(c'^2 + d'^2 + c'd') = 0.$$

Simplifying by xp we get

$$xpm^2 + xpb'^2 + b'\left((1+x)^k + xpm\right) + p(c'^2 + d'^2 + c'd') = 0.$$

Reducing this equation modulo p we get $b'\left((1+x)^k + xpm\right) \equiv 0$ which implies $b' = pb''$ for some $b'' \in \mathbb{F}_p$. The norm equation becomes

$$xpm^2 + xp^3b''^2 + pb''\left((1+x)^k + xpm\right) + p(c'^2 + d'^2 + c'd') = 0.$$

Simplifying again by p we get

$$c'^2 + d'^2 + c'd' = n(b'', m, k) := xm^2 + xp^2b''^2 + b''(1+x)^k + b''xpm.$$

Our approach for step 1 will be to generate random m and b'' (with $x+1 \nmid b''$) until the equation $c'^2 + d'^2 + c'd' = n(b'', m, k)$ has solutions, then to solve this equation for c', d'. As will be clear later, the equation has a solution if and only if all the irreducible factors of n are of even degree. So in particular

- We will choose $b'' = b^{(3)}x + 1$ for some $b^{(3)} \in \mathbb{F}_2[x]$ to avoid an x factor.
- As the term $xp^2b''^2$ is of odd degree, we will make another term of higher even degree, with the following strategy:
  - Choose b'' and m randomly of degree equal to or less than R.

- Choose $k = 2\deg(p) + \deg(b'') + 2 + (\deg(b'') + \epsilon)$ where $\epsilon = 0$ if $\deg(b'')$ is even and $\epsilon = 1$ if $\deg(b'')$ is odd.

If R is large enough we get an n with the desired property after sufficiently many random trials on b'' and m. In our implementation, we chose $R = 10$ which is more than enough for 1024-bit parameters. It remains to show how to solve the equation $c'^2 + d'^2 + c'd' = n$ and to explain the condition on the degrees of irreducible factors of n. We begin with the solution of the equation.

Solving $c^2 + d^2 + cd = n$. It is enough to have an algorithm solving it when n is irreducible. Indeed, if $c_1^2 + d_1^2 + c_1 d_1 = n_1$ and $c_2^2 + d_2^2 + c_2 d_2 = n_2$ then $(c_3, d_3) = (c_1 c_2 + d_1 d_2, c_1 d_2 + c_2 d_1 + d_1 d_2)$ satisfies $c_3^2 + d_3^2 + c_3 d_3 = n_1 n_2$. So suppose n is irreducible of even degree.

We describe a continued fraction algorithm for polynomials over $\mathbb{F}_2$ and then we use it to solve the equation. For a fraction $\xi = \frac{P}{Q}$ where P and Q are polynomials, Let $P = a_0 Q + r_0$ where $\deg r_0 < \deg Q$. Let $Q = a_1 r_0 + r_1$ with $\deg r_1 < \deg r_0$, then recursively for $i = 2, ...,$ define $r_{i-2} = a_i r_{i-1} + r_i$ with $\deg r_i < \deg r_{i-1}$. (This is the Euclidean algorithm applied to the ring $\mathbb{F}_2[x]$). Define $p_0 = a_0$, $q_0 = 1$, $p_1 = a_0 a_1 + 1$, $q_1 = a_1$, and then recursively $p_i = a_i p_{i-1} + p_{i-2}$ and $q_i = a_i q_{i-1} + q_{i-2}$. (The fraction p_i/q_i is the i^{th} truncated continued fraction of P/Q.) We see recursively that $q_i p_{i-1} + q_{i-1} p_i = 1$, so $\frac{p_i}{q_i} + \frac{p_{i-1}}{q_{i-1}} = \frac{1}{q_{i-1} q_i}$ and

$$\frac{P}{Q} = a_0 + \sum_{i=0}^{n} \frac{1}{q_{i+1} q_i}$$

where n is the first i such that $p_i/q_i = P/Q$. Define a "norm" v on quotients of polynomials as follows: $v\left(\frac{a}{b}\right) = \deg a - \deg b$ if $a, b \neq 0$, $v\left(\frac{a}{b}\right) = 0$ if $b = 0$, and $v\left(\frac{a}{b}\right) = -\infty$ if $a = 0$. Note that $v(q_{i+1}) \geq v(q_i)$, $v(p_{i+1}) \geq v(p_i)$, and that

$$v\left(\frac{P}{Q} + a_0 + \sum_{i=0}^{n'-1} \frac{1}{q_{i+1} q_i}\right) = v\left(\sum_{i=n'}^{n} \frac{1}{q_{i+1} q_i}\right) \leq -v(q_{n'+1}) - v(q_{n'})$$

As n has even degree, we can compute $\alpha \in \mathbb{F}_2[x]$ such that $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$ (see next paragraph). We apply a continued fraction expansion to $\xi = \frac{\alpha}{n}$ and let p_i/q_i be the successive approximations. Let j be such that

$$v(q_j) \leq \frac{v(n)}{2} \leq v(q_{j+1}).$$

We have

$$q_j^2 + (q_j \alpha + p_j n)^2 + q_j(q_j \alpha + p_j n) \equiv q_j^2 + q_j^2 \alpha^2 + q_j^2 \alpha \equiv 0 \bmod n.$$

On the other hand, as

$$\deg(q_j \alpha + p_j n) = v(n) + v(q_j) + v\left(\xi + \frac{p_j}{q_j}\right) \leq v(n) + v(q_j) - v(q_j) - v(q_{j+1}) \leq v(n)/2 = \deg(n)/2$$

we have

$$v\left(q_j^2 + (q_j\alpha + p_i n)^2 + q_i(q_j\alpha + p_j n)\right) \leq 2\max\left(\deg(q_j), \deg(q_j\alpha + p_j n)\right) \leq \deg n.$$

Consequently,

$$q_j^2 + (q_j\alpha + p_j n)^2 + q_j(q_j\alpha + p_j n) = n$$

and $(c, d) = (q_j, q_j\alpha + p_j n)$ is a solution to $c^2 + d^2 + cd = n$.

Solutions to $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$. As the map $x \to x^2 + x$ is linear in $\mathbb{F}_2$, solutions to this equation, if there are any, are found easily by writing down then solving a linear system of equations. We conclude the exposition of our algorithm by showing the following lemma.

Lemma 2. *For n irreducible, $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$ has solutions if and only if $d := \deg(n)$ is even.*

($\Rightarrow$) Suppose α satisfies $\alpha^2 + \alpha + 1 \equiv 0 \bmod n$. Then $1 = \alpha + \alpha^2$. Squaring each side we get $1 = \alpha^2 + \alpha^{2^2}$, then squaring again and again we get $1 = \alpha^{2^2} + \alpha^{2^3}$,... until $1 = \alpha^{2^d} + \alpha^{2^{d-1}} = \alpha + \alpha^{2^{d-1}}$. Summing up these equations we get $d = 0$, so d must be even.
($\Leftarrow$) Now suppose d is even. Let β be a generator of $\mathbb{F}_{2^d}^*$ and let $\alpha = \beta^{\frac{2^d-1}{3}}$. Then $\alpha^3 = 1$ and $\alpha \neq 1$ so $\alpha^2 + \alpha + 1 = 0$. □

Runtime analysis We give some estimates for the complexity of our algorithm. Assuming the polynomial n generated from random (b'', m) behaves like random polynomials of degree k, the number of its irreducible factors is asymptotically $K = \mathcal{O}(\log \deg n)$ [5]. For n of degree even, we can reasonably approximate the probability that all its factors are of even degree by $(1/2)^K$, hence we will need $2^K = \mathcal{O}(\log n) = \mathcal{O}(\deg p)$ random trials. The factorization of n can be done in $\mathcal{O}(\log^{2+\epsilon} n)$ [11] and the continued fraction algorithm is of complexity $\mathcal{O}(\deg n)$, so the global complexity of our algorithm is probabilistic polynomial time in $\deg p$. Our implementation of the algorithm finds collisions for 1024-bit parameters in a few seconds on a Pentium Intel M processor 1.73GHz.

6 Discussion and Further Work

In this paper, we presented efficient algorithms finding preimages for the LPS hash function and collisions for the Morgenstern hash function with $q = 2$. Similar algorithms with the same complexity can be derived for finding preimages for the Morgenstern hash function and for different q values. Our algorithms build upon the Zémor and Tillich algorithm [16] although they are not trivial extensions of it.

The modified version of LPS hashes proposed by Zémor and Tillich remains unbroken so far regarding both the collision and the preimage properties, as well as their original scheme [12] (with carefully chosen parameters) that used as graph generators $A_0 = \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix}$ and $A_1 = \begin{pmatrix} x & x+1 \\ 1 & 1 \end{pmatrix}$. To avoid our new attack, we suggest

modifying the Morgenstern hash function as follows: multiply by g_0g_1 if the bit is 0 and by g_0g_2 if the bit is 1. However, it is not clear what would be the advantages of such a scheme compared to Zémor and Tillich's, as it would not necessarily have better expansion properties, and comparing the graph generators, it will certainly be slower.

In further work, we would like to study the applicability of our algorithm to the Zémor-Tillich (ZT) hash function. The Cayley graphs used in ZT hashes can be naturally embedded into Morgenstern graphs, so our cryptanalysis of Morgenstern hashes might actually open new perspectives on breaking the ZT scheme. Our results may also have applications outside the cryptographic community. The preimage finding algorithm actually solves the diophantine equation (1) which at first sight seems to be a very hard problem. Our path-finding and Zémor and Tillich cycle-finding may improve understanding of LPS graphs when considering their Ihara Zeta-function. Finally, expander graphs have numerous applications in computer science [6], some of which could benefit from our new path-finding algorithm.

Because of all these actual and potential applications, we stress that our algorithms and their running time estimates still may and should be improved in many ways. The algorithm of Section 4 gives paths of length about $8 \log p$ while the diameter of LPS graphs is known to be $2 \log p$. Choosing a smaller k value in the algorithm will decrease this length and may also improve the running time. Finding other decompositions with less than 4 diagonal matrices is another interesting approach. Finally, adapting our algorithms to make them deterministic is a particularly interesting open problem.

References

1. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
2. FIPS 180-2 secure hash standard.
3. D. X. Charles, E. Z. Goren, and K. E. Lauter. Cryptographic hash functions from expander graphs. To appear in *Journal of Cryptology*.
4. S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.
5. P. Flajolet and M. Soria. Gaussian limiting distributions for the number of components in combinatorial structures. *J. Comb. Theory Ser. A*, 53(2):165–182, 1990.
6. S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.
7. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.
8. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably secure FFT hashing. In *NIST 2nd Cryptogaphic Hash Workshop*, 2006.
9. M. Morgenstern. Existence and explicit construction of $q+1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory*, B 62:44–62, 1994.
10. C. Petit, K. E. Lauter, and J.-J. Quisquater. Cayley hashes: A class of efficient graph-based hash functions. Preprint, 2007.

11. V. Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inf. Process. Lett.*, 33(5):261–267, 1990.
12. J.-P. Tillich and G. Zémor. Hashing with SL_2. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994.
13. J.-P. Tillich and G. Zémor. Group-theoretic hash functions. In *Proceedings of the First French-Israeli Workshop on Algebraic Coding*, pages 90–110, London, UK, 1993. Springer-Verlag.
14. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
15. G. Zémor. Hash functions and Cayley graphs. *Des. Codes Cryptography*, 4(4):381–394, 1994.
16. G. Zémor and J.-P. Tillich. Collisions for the LPS expander graph hash function. To appear in the proceedings of *Advances in Cryptology - EUROCRYPT 2008*, 2008.

A Toy example of the preimage-finding (path-finding) algorithm in the LPS graph

As an example of our preimage algorithm, we now give a second preimage for the message $m =$"This is not for NIST", when the parameters are $p = 1125899906842769$ and $l = 5$. The ASCII code for m is 01010100 01101000 01101001 01110011 00100000 01101001 01110011 00100000 01101110 01101111 01110100 00100000 01100110 01101111 01110010 00100000 01001110 01001001 01010011 01010100 which in base 5 gives 3023231443000032312104001244030134210403244201222121334313104424320 21. We start at the identity, with g_{IV} the identity and $g_0 = M_1$. We identify the six graph generators

$$M_{\pm 1} = \begin{pmatrix} 1 \pm 2\mathbf{i} & 0 \\ 0 & 1 \mp 2\mathbf{i} \end{pmatrix}, \quad M_{\pm 2} = \begin{pmatrix} 1 & \pm 2 \\ \mp 2 & 1 \end{pmatrix} \quad M_{\pm 3} = \begin{pmatrix} 1 & 2\mathbf{i} \\ 2\mathbf{i} & 1 \end{pmatrix}$$

with their indices. The function π we choose is given in figure A. The hash value obtained is

$$M = \begin{pmatrix} 1113908155375639 & 815055784352014 \\ 485525153198538 & 30164330826615 \end{pmatrix}.$$

	-3	-2	-1	1	2	3
0	-3	3	2	1	-1	-2
1	2	-3	3	2	1	-1
2	-1	-2	-3	3	2	1
3	1	-1	-2	-3	3	2
4	2	1	-1	-2	-3	3

Figure 1. Table for the function π: the table gives the index of the next matrix for a given current matrix and a given base 5 digit.

We apply our path-finding algorithm on M. First, we get a matrix decomposition as in Section 4. After 11 trials, the resulting λ, α, ω, β_1 and β_2 values are

$$\begin{aligned}\lambda &= 1051846637406052\\ \alpha &= 698130975272599\\ \omega &= 846326642296745\\ \beta_1 &= 150389273084944\\ \beta_2 &= 480539407839455.\end{aligned}$$

Then we factorize

$$M_\alpha := \begin{pmatrix}1 & 0\\ 0 & \alpha\end{pmatrix} = \begin{pmatrix}349065487636300 + 7952855976122 50\mathbf{i} & 0\\ 0 & 349065487636300 - 7952855976122 50\mathbf{i}\end{pmatrix}.$$

We choose $k = 48$, resulting in $\lambda = 222458048101540$ and $m = 1121038768144160066886982393688699301560731956564062 5$. After 234 random trials for x, we finally get $x = 523712450310834$, $w = 207632734870715$, and $n = 4.248920597612852537218312864980332096 1$. The Euclidean algorithm gives us the solution $y = 2782001231666122912$, $z = 1489057773063985790$. So the lift of M_α is

$$M'_\alpha = \left(\begin{array}{c|c} 311426103887630914544037511835 & 313225492756935640601527301242332 8\\ +\mathbf{i}766565480745454184887163124346 & +\mathbf{i}167653000797624266329369798025251 0\\ \hline -313225492756935640601527301242332 8 & 311426103887630914544037511835\\ +\mathbf{i}167653000797624266329369798025251 0 & -\mathbf{i}766565480745454184887163124346\end{array}\right).$$

We multiply M'_α by each of the lifts of the graph generators. Since $M'_\alpha g'_3$ is divisible by $l = 5$, g'_{-3} is the last (right-hand) factor of M'_α. After $2k$ steps, we get the whole factorization of M'_α, which we translate into a factorization of M_α whose indices are 3 -1 2 2 3 1 1 3 1 3 3 3 2 2 3 -1 2 1 1 -3 1 1 1 3 -1 2 -3 2 3 1 -2 -2 -2 1 2 1 1 -3 2 1 2 1 -2 3 -1 3 2 -3 -2 3 1 -2 3 3 2 -3 -1 2 2 2 -1 -3 -1 -3 2 3 1 2 -3 -1 3 2 2 1 3 -2 -3 1 3 -2 -1 -2 3 1 3 2 1 -2 -1 -1 -3 2 1 1 -2 -3. We get the factorizations of M_ω, M_{β_1} and M_{β_2} the same way. Finally, we put all the pieces of information together and get the sequence -3 -2 1 1 2 -3 -1 -1 -2 1 2 3 1 3 -2 -1 -2 3 1 -3 -2 3 1 2 2 3 -1 -3 2 1 3 2 - 3 -1 -3 -1 2 2 2 -1 -3 2 3 3 -2 1 3 -2 -3 2 3 -1 3 -2 1 2 1 2 -3 1 1 2 1 -2 -2 - 2 1 3 2 -3 2 -1 3 1 1 1 -3 1 1 2 -1 3 2 2 3 3 3 1 3 1 1 3 2 2 -1 3 2 3 -1 -3 -2 -2 1 3 2 -3 -3 2 3 -2 -3 -3 -2 -1 -2 3 1 1 2 3 2 1 1 -3 1 2 2 1 -2 1 1 2 -3 -2 3 -2 -3 1 1 3 1 2 1 -3 -1 -3 -3 -1 2 3 1 -3 -3 -1 -1 2 -1 3 -2 1 -3 -3 -1 -2 1 1 -2 -1 -1 3 -2 3 2 2 1 -3 -2 -1 -3 -1 -3 -1 3 2 -1 3 3 -2 -1 -2 1 1 2 2 -3 -1 3 2 2 -3 -2 -3 -1 -3 1 -3 -2 -1 3 1 -2 3 -2 3 2 1 3 -2 -2 -3 -3 -2 -2 3 -2 -2 -3 -1 3 1 3 -1 -3 -3 -3 -3 -2 1 3 3 1 -2 3 -1 -2 1 2 -3 1 -2 -2 1 -2 -2 -1 2 2 2 2 2 1 -3 1 1 2 1 1 3 3 -1 3 3 -2 -1 3 1 2 -1 2 3 -1 2 -3 -2 1 -2 1 1 3 -2 2 -2 -3 -2 -1 3 3 -2 -1 3 2 -3 2 3 -2 1 1 1 1 -2 1 1 2 -1 3 -1 -2 -1 -2 -1 -2 -2 3 2 1 3 - 2 -2 -3 -3 -2 3 3 2 1 1 1 1 3 2 -1 -3 2 -3 -2 -1 -3 1 -2 -2 -2 1 -2 -1 -2 -1 2 - 1 -3 -1 -3 -2 -1 -2 -3 -1 -3 -1 2 2 3 3 -2 -2 -2 -3 -1 -1 -1 2 -3 -1 3 -1 2 -3 - 3 that collides with the original message “This is not for NIST”.

B Second preimage of "This is not for NIST" for LPS hashes with 1024-bit parameters

Now we repeat what we did in Appendix A, this time for
$p = 17976931348623159077293051907890247336179769789423065727343008115773267580550096313270847732240753602112011387987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413932 9$,
which has 1024 bits. The sequence we get is

3 1 3 3 2 3 2 3 -2 -1 -2 1 -3 1 2 -1 -3 -2 -3 -1 2 -1 -1 -3
-2 -2 -3 1 2 -3 -3 2 -1 3 2 2 2 -3 -3 1 1 2 -1 3 -2 -2 1 1 -2 -3 2 -3 -3 1 3 1 3 -2 3 2 3 2 3 3 -2 1 1 -2 3 1 3 -2 1 3 1 -3 1 3 3 1
2 2 1 1 1 -2 3 -1 -1 -2 3 2 3 1 3 1 -3 1 1 -3 2 2 1 3 -1 -3 1 2 3 -2 3 2 1 3 2 1 2 3 3 -2 1 2 -1 3 1 3 1 3 -1 3 2 -3 -1 -2 -3 2 -3
-1 -2 3 1 2 -3 -3 -1 -3 -3 -3 -1 -3 -3 1 3 3 3 -2 -1 -2 -2 3 3 1 -2 -2 -2 1 -2 3 1 -3 2 -1 2 3 3 2 1 1 -3 -2 -3 1 -3 -3 -2 -1 -1 -2
-2 -3 -2 -1 2 -3 -3 -3 1 2 -3 -2 -1 2 1 3 3 -2 1 -3 2 3 -2 3 1 -3 -2 -1 3 3 1 3 -1 -2 1 2 -3 -1 3 3 -1 3 -1 -2 3 3 2 2 -3 2 1 2 -1 2
2 -3 -2 -2 1 -3 -3 1 2 2 -3 -1 -3 1 2 -3 2 2 -3 -3 -1 -1 -1 2 1 2 3 -1 -1 3 2 2 -3 -2 1 3 -2 3 -1 3 -2 -2 1 2 1 1 2 -1 -1 -1 3 -1 2
2 2 -1 -3 -3 -3 1 -3 -1 -1 -1 3 3 3 3 -2 -2 3 -1 3 2 -3 1 2 2 -1 -1 3 -2 3 -2 -1 3 -2 3 -2 3 -2 -1 2 1 3 1 -2 3 2 2 3 -1 -1 3 2 2 3
-2 3 -1 -3 2 3 -2 3 1 -3 - 3 -2 -2 1 2 1 3 1 -3 -2 -1 2 2 2 -1 3 3 2 2 2 2 3 3 -1 -1 -1 -3 -2 3 1 2 -1 -2 1 -3 -1 3 -2 -3 1 -3 -2 -2
-1 -1 -1 -1 -3 -3 2 2 1 -3 -3 -2 -3 -1 -3 2 -3 1 -2 1 1 -3 -1 -3 -2 -1 -3 -1 -1 3 3 1 3 -2 3 -2 -3 -1 2 2 3 -1 3 -2 -3 1 3 3 2 2 3 1
1 -3 -1 -3 -1 -2 -2 -1 -3 -3 1 -2 -2 -1 3 3 1 -3 -1 -1 -3 2 3 -1 3 -2 -3 2 -3 -2 1 -3 -1 -1 -2 3 -2 -2 -3 1 -2 -2 -2 3 -1 2 3 -1 2 3
-1 -2 -1 3 -2 1 2 2 3 1 2 -1 2 1 -3 -3 -2 3 3 1 2 -3 1 2 1 -2 -1 2 -3 1 2 -3 -3 -1 -1 -3 2 1 3 -1 2 3 3 -2 - 3 1 3 1 -3 -1 -3 2 -1
-3 -3 -1 3 3 -2 -1 -2 1 3 -2 -3 2 1 -2 -1 3 -2 1 2 -1 2 3 1 -2 -3 1 -2 -2 -1 -2 -2 -2 3 2 1 2 -3 -2 -1 -3 1 -3 -2 -2 -1 -2 3 -2 -2 -1
-3 - 3 1 -3 1 3 2 -3 -2 3 3 1 -3 -2 1 3 2 1 3 -2 3 -1 2 -3 -3 2 3 1 -2 -1 -3 -3 1 3 - 2 -3 -2 1 2 1 -3 -1 -2 -3 -3 1 2 3 2 1 1 -3 -3
-2 3 3 3 2 -3 1 -2 1 -2 3 -2 1 3 3 3 2 1 3 1 3 2 1 1 2 -1 -2 1 3 -1 -3 -1 3 3 2 -3 1 3 1 -3 -2 -1 2 2 -3 1 3 3 3 -2 3 3 1 1 3 -2 1
1 1 2 1 -3 -3 -3 -2 -1 -2 -1 3 3 2 -3 1 2 1 -3 1 -3 -3 -2 -2 3 2 2 3 -2 -2 -1 2 -3 -3 2 2 -3 -3 2 -1 -1 -3 2 -3 1 2 1 3 -2 -3 1 -2 1
3 2 -3 -3 -3 -2 -2 1 2 -3 1 2 -1 -1 -3 -2 3 2 2 2 -1 -3 -1 2 1 -2 -2 3 3 2 2 3 -1 -1 -2 - 3 2 3 -2 1 1 -2 -1 -1 -2 1 1 -3 -2 1 1 1
-3 -3 -3 2 2 1 -2 1 -2 -3 1 -3 -1 -2 3 -2 3 1 -3 -1 2 -1 2 -1 -3 1 1 -3 -1 -3 2 3 1 -3 -1 2 1 2 -3 1 -3 -2 1 -2 -1 -3 -2 -1 -2 -3
-1 -1 -2 1 -2 1 3 1 3 -2 3 1 3 3 3 3 -1 -2 -3 -1 3 3 -2 -2 -1 -1 3 -1 -2 3 1 -3 1 3 2 3 -2 -2 -3 -2 -2 -1 -2 -3 -1 3 2 1 2 -3 1 2 -3
-2 1 3 3 3 -1 3 2 -1 -2 1 -3 -1 -2 -2 -3 2 1 1 2 2 2 1 2 -1 -2 1 1 2 2 3 -2 1 3 3 -2 -1 -3 2 3 3 -2 -1 2 -3 1 1 1 -2 -2 -3 1 -2 3 -1
3 2 3 3 -2 1 2 1 -3 -2 -2 -2 -1 3 -1 -3 - 3 -2 -1 -3 1 -3 2 -1 2 1 2 1 3 3 3 -1 -1 -2 1 2 -1 -1 3 -2 1 2 3 -1 3 -1 2 -3 -2 -3 1 -2
-3 2 3 3 2 -1 -2 1 -2 -1 -3 -2 -1 -2 3 -1 3 2 -3 -3 -1 3 1 -3 2 -1 -1 2 -3 -3 1 -2 1 -3 -1 2 3 1 1 -2 -3 -2 3 -1 -1 -1 -3 -1 -1 -2 -2
-3 -3 1 -2 -3 2 2 3 3 2 1 3 2 -3 -2 -2 1 2 2 3 -2 3 2 -1 3 3 1 2 3 -2 3 -2 1 -3 1 3 2 1 1 2 2 -1 2 1 2 3 1 -2 1 1 3 -2 1 -2 -3 -2 3
1 -3 -3 1 1 2 1 -2 -2 3 -1 2 -1 -2 -3 -2 1 3 3 1 -3 2 -1 2 -3 2 3 1 1 1 3 2 3 -2 1 -2 1 -2 1 3 -2 -1 2 -3 -1 3 2 -3 1 -2 -1 - 2 -2
-1 2 1 -3 -1 2 -1 -3 2 -1 -1 2 2 -3 -3 1 2 1 3 -1 3 1 2 2 -3 2 -1 2 2 1 1 - 3 -1 -1 -1 3 3 -2 -2 3 -1 -2 -1 3 1 3 3 -2 1 -3 -2 -2 1
-2 1 -2 -1 -3 2 2 -1 -2 -1 -2 -1 -3 -2 -1 -1 2 2 2 2 -3 -2 -2 -2 1 3 1 -2 -3 -1 -1 -2 3 2 3 -1 3 2 -1 -2 3 -1 2 -3 -2 -2 -2 3 -1 3 2
-1 2 2 -1 -3 1 -3 -3 2 1 -2 -1 -2 1 -2 -2 1 2 1 -3 -3 -1 -1 2 -3 -2 -3 1 -2 1 1 1 -2 -2 -1 -3 1 2 1 3 -2 -1 -2 3 2 2 2 -3 1 3 1
3 -2 3 -2 -2 -3 -2 -3 -3 2 3 -2 1 1 3 1 -2 -3 -2 -2 1 2 -3 -2 -2 -1 -1 2 -1 -2 3 2 3 -1 -2 3 2 3 -2 -2 -2 1 2 1 -2 3 2 3 -1 -3 -1 -1
3 -1 -2 1 -2 1 2 3 -1 3 1 2 -1 2 1 1 2 3 2 -3 -3 -3 -2 -3 -3 2 -3 -1 3 3 2 2 1 -3 -2 -3 -3 -1 3 -2 -2 -1 3 -1 -3 2 -3 2 2 -3 2 3 3 2
-1 -2 3 3 2 3 2 2 1 -3 2 1 2 2 2 2 -3 -2 1 -3 1 2 3 3 3 3 -2 -2 1 2 -3 -3 2 3 3 2 -1 -1 3 2 2 2 -1 2 -1 -2 3 2 1 1 -2 -2 -2 3 1 1 3
-2 1 2 2 -3 2 1 1 -3 -3 2 2 -1 -2 -3 -1 -3 -3 -3 -1 -1 -2 1 -3 1 -3 -1 -1 -3 -1 -3 1 2 -3 2 1 -3 2 -3 -2 3 2 -3 1 3 1 -2 -3 -3 -1 -3
-2 -3 2 2 2 3 1 1 2 1 1 1 2 2 -1 -2 -2 3 -1 3 -1 -2 3 -1 -3 1 3 3 1 -3 -2 -1 -3 -1 2 3 -2 -1 -2 -1 -3 -1 -3 -1 2 1 3 -2 -2 3 2 1 -2
3 -1 3 1 -2 -1 -1 -3 -2 1 -3 1 -3 -2 -1 3 -1 -2 -1 3 3 2 3 -1 2 1 -3 2 2 -1 -2 -2 -2 3 3 3 -1 -3 -1 -2 3 -1 -1 3 2 1 1 3 3 3 3 1 2 3
2 -3 -1 -1 3 -2 3 2 -1 -2 -3 1 1 -3 -2 -3 -3 1 -3 2 3 -1 -3 2 -3 2 -1 3 1 1 -3 1 - 2 3 2 3 1 3 -1 2 3 -1 2 -3 -2 -2 -1 3 -2 -2 3 2 3
-2 -3 1 3 2 1 3 3 3 -2 -1 2 -1 3 -1 3 -2 1 1 -3 2 3 2 3 -2 -3 -2 -3 -3 -3 -1 -3 -3 -3 -3 2 -1 2 -1 3 -2 1 1 -3 -1 -3 -2 -1 -1 -3 -1 2
2 2 -1 -1 -2 1 2 3 -2 -3 2 2 -1 3 3 -2 -1 -3 -2 -3 1 -3 -1 2 1 2 1 2 2 -1 3 2 2 -3 -3 1 3 -1 2 1 3 2 -3 -3 1 -2 1 1 2 -1 2 2 -1 2 1
1 3 1 -2 1 -2 -1 -3 -1 -2 -1 -3 1 3 3 1 2 3 1 1 -2 -2 1 -2 -2 -1 -3 1 3 1 -3 -3 -3 - 1 -1 3 -1 -1 -3 2 -3 -3 -1 -2 -1 -2 1 1 2 2 2 1
2 2 -1 2 1 3 3 2 3 3 -1 -3 -1 -2 1 3 1 -2 1 -2 1 -3 -3 -2 1 1 -3 2 2 2 3 1 3 2 2 2 -3 2 -1 -3 1 1 -3 1 -2 3 1 2 -3 -3 1 1 -2 -3 2

-1 3 3 -1 -1 -1 -3 1 -3 -1 2 -1 -1 -1 -1 -3 -3 -2 3 1 1 1 -2 - 1 2 3 2 1 -3 1 1 3 -2 -3 2 -3 -2 -2 3 3 1 3 2 -1 3 1 3 -1 -3 -3 1 1 2
-1 -3 -3 1 3 -2 1 -3 2 2 -3 -3 -2 3 2 -3 2 3 3 2 -1 -3 2 1 2 -3 -3 1 -3 2 3 1 -3 -2 -2 -3 -2 -1 -3 2 -1 -2 -3 -2 -1 3 -2 3 -2 -3 -3 2
2 2 1 2 2 -1 -3 -1 -3 -1 -1 2 2 3 1 -2 3 -1 -2 1 3 1 2 1 3 -2 -2 -2 3 2 2 -1 -2 -1 -3 2 -3 -3 -2 3 3 -1 2 2 -1 3 2 - 3 1 3 -2 1 2 2
-1 -2 -2 -2 -2 -3 2 -3 -2 3 -2 3 3 -1 -2 1 -2 3 3 1 -3 2 1 2 2 -1 3 -1 -3 -1 3 -1 -1 3 1 -3 1 -3 2 2 2 -1 -1 2 2 -3 -3 1 -3 1 -3 -1
-3 1 -3 -2 -2 -3 -3 -2 -1 -2 -3 -3 1 -2 -3 -3 -1 -1 -3 -3 -1 -1 2 1 1 -2 -1 2 2 2 1 -3 2 -1 3 1 1 -3 -1 3 -2 3 2 -3 2 1 -2 3 3 -2 -2
3 3 -1 3 3 -1 3 3 1 1 -2 -1 -1 3 -1 -1 2 -3 -1 -3 2 -3 -3 2 1 2 2 -3 1 3 -1 -3 -1 -3 -2 -1 -3 -3 2 1 2 -3 2 -3 1 2 2 -3 -3 -1 2 2 3
1 -3 -1 -2 3 -2 1 1 -2 1 1 -3 -3 2 -3 -1 -2 -2 -2 -1 -1 -3 -3 1 3 1 1 3 2 3 1 1 -3 -3 -1 -3 -2 -3 -3 2 -1 -1 -3 2 -3 -1 -3 1 -2 -1 3
1 1 -3 1 1 3 -2 -2 -3 -1 -2 3 3 3 3 -2 -3 -3 1 2 2 1 1 1 -2 3 3 -2 3 -1 -3 2 -1 2 3 1 3 2 2 1 2 3 1 1 -3 -3 -1 2 -3 -3 -1 -1 2 -1 2
3 2 1 2 2 3 1 1 1 3 -1 -3 -1 3 2 3 -1 -1 -2 3 1 -3 2 1 2 3 -2 1 1 -2 -1 -3 -1 3 2 2 3 -1 -3 1 -3 -2 -2 3 -1 -1 -1 3 3 -2 3 1 -3 -2 3
-2 -1 3 -1 2 1 -3 -3 -3 2 -1 3 3 -2 3 2 1 3 -2 -3 -2 -3 2 2 1 3 1 -2 1 2 3 1 -2 -3 -2 -3 2 2 2 -1 -3 -3 -3 -1 -2 3 3 2 2 -3 2 3 -2 1
-2 -2 3 2 -3 -1 - 3 -1 -1 2 2 2 1 1 2 1 3 3 2 -3 1 1 -3 -3 -2 1 3 3 -2 3 -2 -1 -1 3 3 3 -2 -3 1 1 -3 -3 2 -1 -2 -2 -2 -1 2 1 -3 1 -2
-1 -2 -2 1 3 3 1 2 2 2 2 1 -3 2 2 3 -1 -3 -1 3 -2 1 -3 -2 -1 -1 3 -2 1 -2 1 3 1 -3 1 3 1 3 -2 1 2 3 -1 2 -1 -3 -2 1 3 1 3 1 3 2 -3
1 1 3 1 -3 -2 -3 -3 2 2 1 1 3 -2 3 1 3 3 -1 -1 3 1 -2 -2 1 1 -3 -2 -2 -1 - 2 -3 -3 1 -2 3 2 2 -1 -1 2 -1 3 2 -1 3 1 -3 -2 1 -2 -3 2
-1 -1 2 -3 2 -3 1 3 1 3 -1 2 3 2 2 -3 1 2 2 2 2 2 3 -2 1 2 -3 -2 -1 -2 3 -1 -2 -3 -2 1 2 -3 2 -1 -2 -2 3 3 -2 -3 -3 -3 -2 -3 -2 -2 1
3 -2 3 -1 -3 -1 -1 3 2 2 3 1 -2 3 3 3 -1 -1 3 3 -2 -1 -1 3 3 -1 2 1 2 -1 2 -1 -3 1 2 1 -3 -1 3 -1 -2 1 2 -3 -2 -3 -1 -2 1 -2 -2 1 2
-3 2 1 2 -3 -3 -1 -2 -3 -1 2 -1 -2 -3 -3 1 -3 -2 1 2 3 1 2 3 1 -3 -1 2 -3 1 -3 1 1 1 3 -1 -1 -2 -3 2 2 -1 -1 -3 -1 3 -1 -1 3 3 -2 3
-2 -1 -3 -2 -1 -3 -3 1 1 2 2 -3 1 2 1 3 -2 1 -3 2 1 -3 -3 -3 2 3 1 3 3 -1 3 1 3 1 1 1 2 3 -2 -3 1 -2 3 -2 -3 -3 -3 -3 1 3 1 -2 -2 -3
-1 3 -1 -1 -2 -2 1 -2 -3 2 -3 -2 3 -2 -1 -2 -2 3 -2 - 3 -3 -1 -3 2 -1 3 -2 -1 -3 -2 -1 -2 -3 1 1 3 3 -1 3 -2 -3 -3 -2 -2 -2 3 2 -1 -2
3 1 3 -2 1 -2 3 1 1 3 -2 1 3 -1 -3 -3 1 -2 -2 3 -1 -2 -2 1 2 1 -2 -1 2 1 -2 1 -2 -3 -2 -3 1 2 -1 -2 -2 -3 -3 -2 -3 1 1 1 3 1 -3 -3
-1 -3 1 -3 -3 -2 -2 -1 -1 3 1 -2 -1 -1 2 3 -2 1 3 -1 -3 -2 -2 -2 3 -1 -3 -3 -3 -1 -1 -3 2 1 3 2 -3 -1 -1 -3 - 2 -1 3 -2 -3 1 2 2 -3 -3
2 -3 -3 1 -2 -2 -1 -1 3 -2 -1 3 -2 -2 -3 2 -3 -1 -1 -1 -2 -1 -2 -3 2 1 3 -1 -3 -1 3 1 -3 -1 -2 -1 -2 -3 2 1 2 1 1 2 2 3 1 2 -1 -3 1 1
3 2 1 2 3 2 -1 3 2 2 3 -1 -3 -2 -3 -1 2 -3 -3 2 -1 2 -1 -1 -2 -2 3 2 2 1 2 -3 -3 -3 -2 -2 -1 2 -3 -1 2 1 -2 1 -2 -3 -1 3 -2 1 3 1 1
1 1 -3 1 -3 -3 1 3 -1 -3 2 -1 3 2 2 1 1 -2 -2 -3 2 -3 1 -2 3 -1 -3 1 1 -2 -2 3 3 -1 2 1 -3 -2 3 -1 -2 3 3 -1 -2 -2 -1 -3 2 1 -3 2 -1
-3 2 -3 -2 1 2 -1 2 1 2 -1 -2 -3 -3 -1 -3 2 3 -2 -2 1 1 -3 2 -3 -3 1 -3 -3 2 -1 2 -3 -1 -2 1 1 -3 1 3 3 -1 2 3 1 3 3 2 3 3 -1 -3 -2
-3 - 3 -3 1 2 3 -1 -2 -3 -3 1 -3 2 -1 3 1 2 1 -3 -2 -1 -1 -3 -2 -2 -2 3 3 -1 2 2 3 1 -2 -3 -1 -2 1 3 1 2 -3 -3 -1 3 -2 1 -2 -2 -1 2
-3 -1 2 -3 -2 1 -2 -3 -1 -1 -1 -3 -1 -3 -2 -3 2 -3 -2 3 2 1 -3 -3 1 2 1 2 -3 1 -2 1 2 -3 1 3 1 -2 -1 -2 -2 -2 -3 -2 1 -2 3 1 1 3 2 -1
-3 -3 -2 -1 -1 2 -3 -1 3 -1 3 -1 -1 2 1 -2 1 3 1 -3 -2 1 1 -2 1 2 3 2 -3 1 2 -1 2 -3 -3 -3 2 -3 -1 -3 -2 -1 2 3 3 3 3 3 -1 -1 2 -1
-2 -3 -1 -2 1 1 2 1 1 1 1 2 3 3 1 -2 -2 1 2 1 3 2 -3 -1 -1 -2 1 1 3 2 -3 -2 3 1 -3 1 -2 -2 -2 3 3 -2 1 -2 1 -2 -2 -2 -3 -2 -2 -1 2 1
3 -2 -3 -1 -2 3 -1 -2 -1 2 -3 -1 3 -2 -3 2 1 -2 3 -1 -3 -1 2 3 -1 -2 1 2 -1 2 -3 -2 -3 1 3 -2 1 -3 2 3 3 -1 -1 -3 2 -1 -1 2 1 3 3 1
1 -2 -3 -3 -1 2 3 2 3 -1 -1 -2 3 -1 3 2 -1 -1 2 1 -2 3 2 -3 2 - 1 -3 -3 -2 1 3 -2 -2 3 -2 3 -2 1 2 2 -3 2 3 2 3 -1 3 2 -3 1 -3 1 3
-2 -2 -3 -1 - 3 1 3 1 -3 -2 1 -2 -3 2 3 3 -2 -1 2 1 -2 -2 -3 -2 -1 -2 3 -2 1 1 -2 -1 2 -1 2 1 -3 -2 3 1 -3 -2 -1 -1 -2 1 2 -1 -1 3 2
-3 -3 -3 -2 3 1 3 -2 -2 -1 -2 3 -1 -2 -2 3 3 -1 -3 -1 -3 2 -1 3 3 -2 3 -1 3 -1 -2 1 -3 -2 1 1 -2 -1 -3 1 3 -1 -2 -1 -1 -1 2 3 -1 3 -2
2 3 3 -1 -2 -2 -1 3 1 -3 2 -1 -1 -1 -1 3 3 -2 -3 -3 1 3 2 -1 -3 2 - 1 -3 -2 -3 -2 -1 -1 -2 3 -2 -3 -2 -2 -1 2 -3 -1 -1 2 -3 2 3 3 2
-3 2 3 -2 -3 -3 -3 -2 -3 -1 -2 -2 3 -2 1 -3 1 -3 -2 -2 3 2 -3 1 -2 -1 2 -3 2 -1 -3 -1 -3 2 -1 2 2 2 -1 2 1 3 1 3 1 1 1 2 -3 -1 -2 1
1 3 -1 -2 -2 -2 3 -2 -2 3 2 1 1 3 -1 -1 3 3 -1 -1 -2 3 2 2 -3 1 -2 -2 -1 -1 2 -3 -1 -3 -2 1 -3 2 2 1 2 -1 -3 1 2 -3 1 -3 -1 -1 -1 3
3 3 -1 -1 -1 2 2 1 2 2 2 3 -2 -3 -1 3 1 -3 -3 -2 -3 1 -2 -1 -1 2 3 -1 -3 -3 -3 -1 -2 -3 -2 3 3 -1 -1 -3 1 3 2 1 3 1 -3 2 3 -2 3 1 2
2 2 3 3 -1 -3 -1 -1 -3 -2 1 -2 3 -2 3 3 2 -1 3 2 1 -2 -2 1 1 1 2 3 3 2 1 3 1 1 -2 -3 -2 -1 2 -1 -1 3 2 -3 -1 -2 -3 -2 -1 -3 -3 -3 1
3 2 -3 -2 1 -2 -3 2 1 3 1 -2 1 1 2 1 3 1 -2 -2 1 2 -3 1 1 -2 -2 3 3 -2 1 2 3 -2 1 1 3 1 2 -1 -1 -1 -1 2 3 -1 -3 2 2 -3 2 2 2 3 - 1
-3 -1 -2 1 1 -2 1 3 3 -2 3 1 3 -1 -2 1 2 -3 1 1 1 1 3 -2 -2 1 -2 -1 2 -3 1 -3 2 -1 2 2 -1 -2 3 3 -2 3 -1 -1 2 1 3 -1 2 1 -3 -1 -1 3
1 3 -1 2 2 -3 -1 -2 -1 3 3 -2 1 3 -1 -2 -3 1 2 2 -3 -3 -2 -1 -3 1 3 2 2 -3 -2 -3 1 2 2 -3 -3 1 -2 1 -3 -3 2 -3 -3 1 1 2 1 -2 3 1 3
-1 3 -2 3 -2 3 3 3 -1 2 -1 -1 -3 1 2 1 3 -1 -3 -2 -3 1 3 1 -2 -3 -3 -2 1 2 -1 -3 -1 -2 -1 -2 1 -2 -3 -3 -1 3 3 -1 2 -3 1 -3 -3 -3 1
-3 2 2 1 -3 -2 -3 -1 -3 -3 1 3 -1 -1 -3 -2 -2 -3 -1 -1 3 1 2 2 -1 -2 -3 -1 2 3 -2 - 3 -1 3 -1 -3 2 1 -2 -2 1 -3 -2 1 1 3 2 -3 2 3 -1
3 -1 -2 -2 3 1 -3 -2 -2 -2 1 3 1 3 -1 2 2 3 3 1 2 1 -3 -1 3 -2 1 -3 -3 1 1 3 -1 3 1 -2 1 2 -1 2 3 3 1 2 2 3 -1 -3 2 3 2 2 1 3 2 2
-3 -3 2 1 1 3 3 3 1 1 -2 -3 2 3 2 -3 -1 -1 2 1 1 -3 2 3 -1 3 1 -2 3 1 -2 1 3 -2 -3 -3 -1 -1 3 -1 -3 -1 3 3 3 1 3 -2 3 -2 -1 -1 -1 3
3 -1 -1 2 1 -2 -3 1 -2 -2 -1 -2 -3 2 1 -2 -3 -2 3 -1 -1 -3 -1 -3 -2 1 3 2 3 -2 -2 -3 -3 - 1 2 2 2 3 3 2 1 1 -2 3 3 1 -3 -2 -3 -2 -1 2

1 3 1 -2 1 3 -1 -3 -2 3 -1 -1 2 2 -3 -3 1 3 1 3 3 1 -3 -3 -1 2 2 -1 -2 -3 1 2 2 -1 2 2 3 -1 -3 -1 2 -3 2 -1 -2 -2 -3 2 -3 -1 -1 -2
-1 -2 -3 2 1 1 2 3 -1 3 1 2 3 -1 2 -1 -3 -2 1 -2 -3 -1 -2 -3 1 2 -3 1 -2 3 3 -2 -1 -3 -2 1 3 -1 -1 -1 3 -1 3 2 -1 -2 -3 1 1 -3 -1 2
-1 -2 1 -2 -3 -3 -1 -2 -2 3 -2 3 2 2 2 2 -1 -2 -3 1 1 3 -1 2 1 -2 3 2 -1 2 2 3 2 -1 -2 -3 1 2 2 -3 -1 -3 1 -3 -3 -1 2 2 3 2 1 3 2 2
1 3 -2 -1 2 2 3 -1 2 -3 2 3 2 -3 -2 -1 2 -1 -2 -2 -3 -2 3 1 1 2 3 2 -1 -2 -3 1 1 3 1 3 2 3 3 -2 -2 -2 -2 3 -1 3 1 1 -2 -2 -1 -1 2 -1
3 1 -2 -1 2 2 2 2 2 -3 1 1 2 3 -1 -1 2 -1 3 1 2 -3 -2 -3 -2 1 -2 -3 1 3 1 -3 -1 3 1 2 -1 -1 2 -3 -1 3 2 -3 -2 -3 2 1 -3 2 1 1 1 3 3
-2 -1 2 3 3 -2 3 2 1 2 1 3 1 -3 -3 1 1 2 -1 -2 -1 -1 -1 -1 -3 -1 -3 -2 -1 -3 -1 -3 -1 -1 2 -1 2 3 1 -2 -2 3 3 1 2 -1 -3 -2 -1 -1 2 -1
3 -1 -3 -3 -3 -1 3 2 -3 -1 3 3 -2 -2 -3 -1 -2 1 -3 1 -2 -1 -2 -3 -1 3 -1 -2 1 3 1 1 1 -3 2 -1 -1 3 3 1 -3 -1 -3 2 2 -3 -3 -3 1 1 3 3
-2 -1 -3 -2 -3 -1 2 3 -1 -3 2 1 -2 -2 3 2 2 1 -3 -2 1 1 -2 3 1 -2 -3 -2 1 -3 1 -2 -3 1 3 2 1 -2 3 -2 1 -3 2 -1 -3 1 2 1 2 -3 -2 1 2
2 -1 -3 -3 2 -3 1 -3 1 -2 -1 -3 -1 -3 2 2 -1 2 3 -2 1 2 3 1 2 -3 1 -3 1 -3 -2 1 -3 -3 -2 1 2 -3 - 2 -1 3 -1 -3 2 -3 -1 -3 -2 1 2 -3 2
3 2 2 -3 -1 3 3 3 2 -3 1 3 2 1 -2 -2 3 1 3 1 2 2 2 1 2 2 -1 -2 1 -2 -2 -1 2 -1 3 -1 3 -1 -2 3 -1 2 -3 -1 2 2 3 -2 1 2 2 -1 2 2 2 -3
-3 -1 3 2 -1 3 -2 -2 -2 -1 -1 3 -1 -2 -1 -2 -2 1 -3 -3 2 3 3 -2 -1 -3 -3 2 -3 -3 1 2 -1 3 1 2 -3 -3 -3 2 -1 2 -1 -2 1 1 -3 -3 2 -1 2
1 -2 1 -2 -1 -2 1 3 -2 -2 -3 -1 2 3 -2 -3 2 3 -1 -2 -2 -2 3 2 2 -1 -3 2 3 -2 -3 1 -2 -1 -2 -2 -3 1 3 3 3 1 3 -2 -1 2 2 1 3 2 -1 2 -3
-2 -1 3 3 1 1 -2 -3 -1 -2 3 2 -3 -1 2 3 1 3 3 -1 -3 1 1 2 -1 2 2 2 1 3 -2 3 -1 -3 -3 1 3 -1 2 -3 -2 3 1 -3 -3 -2 -2 -3 -1 3 1 3 -1
-2 -1 2 1 1 1 -2 3 -1 -1 2 2 3 -1 -1 -3 1 -3 -2 1 1 -2 -1 3 3 1 1 3 -1 -2 1 2 -3 1 -3 -2 -1 -3 -1 -3 -2 -2 3 1 3 2 3 -2 -2 3 -1 -1
-2 1 1 3 -2 -3 -3 -1 2 3 -2 1 2 -1 -2 -1 -3 1 -2 1 -3 -3 -1 2 -1 2 1 3 2 3 -1 2 2 -1 -3 2 -3 1 3 2 -1 -2 -1 3 3 -1 2 1 -3 1 -2 3 3 3
2 -1 -2 -1 2 3 -1 3 1 3 -1 -2 3 2 -1 2 1 3 -1 -1 3 2 2 -1 2 -1 3 2 2 2 3 1 2 2 3 3 1 3 1 3 -1 2 2 -3 2 3 1 1 -2 3 2 -1 2 2 -3 1 1
3 1 3 -2 1 3 2 1 -3 -3 -2 1 3 1 3 1 2 3 -1 2 -1 2 -3 -3 -3 1 3 1 1 -3 -3 1 -3 -2 -1 -3 2 -3 -3 -2 1 -2 -3 1 2 -1 -3 1 1 2 1 -3 -1 -2
-3 2 1 2 3 2 -3 -1 -2 3 3 -2 3 -2 -1 -1 -2 1 3 -2 1 3 -2 1 3 3 2 3 -1 3 -2 -2 -3 -2 1 3 2 3 2 -1 -1 2 3 2 3 3 3 1 -2 1 -2 1 -2 -1 -2
-2 -2 1 2 2 1 2 2 -1 3 -1 -1 2 2 1 -3 -1 2 2 -3 -2 3 3 -2 -3 1 2 1 -2 -2 3 1 -3 -1 -3 -2 -1 -3 1 -3 1 2 1 1 -3 -3 1 3 1 3 2 2 1 2 -1
2 -1 2 -3 -3 1 1 1 1 3 1 2 -1 2 -1 3 -1 -3 2 -1 -2 3 -2 -2 1 3 2 -1 -2 -2 -1 -1 -2 -3 -1 2 -1 -2 1 1 -3 -2 -3 1 3 -2 3 2 -1 3 -1 -2
-3 -2 -1 -3 -2 -1 -2 3 3 -1 2 3 -2 -3 2 2 -3 -1 3 -2 1 -3 2 2 1 2 -3 -3 -2 1 1 -3 -2 -1 2 -1 -3 -1 3 1 -2 -2 3 -2 -3 -3 2 1 -2 -3 1 1
-3 1 1 3 -2 2 -2 1 3 2 -1 -2 -3 1 3 2 -1 -1 -1 -2 3 1 -2 -1 2 1 1 1 2 3 2 -1 -3 -3 2 -3 1 -3 -3 -1 -1 2 3 2 2 3 -2 -3 2 2 1 2 -3 2
-1 3 1 -2 -2 -1 -3 2 3 2 -1 -2 -1 -1 -2 -3 2 1 2 -3 2 -1 2 3 -2 3 -1 -2 -3 1 3 -1 -2 -1 -2 3 3 -2 -3 -2 1 2 2 2 2 -3 -2 3 -2 -2 -3 -3
-3 -2 -3 -3 1 -3 -1 -2 -1 3 1 2 -1 2 2 -1 -1 -1 -3 1 -2 3 2 2 -3 -2 -2 -3 -1 2 -1 -1 -3 -2 -2 3 2 -3 2 3 -2 3 - 1 3 -2 3 3 1 -2 -3 -3
1 1 3 -2 1 -3 -2 -2 -1 -2 1 -2 1 1 3 -2 -2 -1 3 -1 -3 -1 - 3 1 -2 -1 -3 1 3 2 1 2 -1 -2 -3 2 2 2 3 1 3 3 1 -3 -2 3 2 -1 -3 -2 -1 2 1
2 1 2 -1 -1 3 1 3 -2 3 -1 -2 -1 -1 -2 3 1 -3 -2 3 1 2 3 1 1 3 3 -1 -3 -2 -1 3 3 3 1 3 3 3 -1 3 -1 2 -3 -3 1 1 -2 -2 -1 -2 -2 3 -2 -2
3 1 -2 -3 -2 3 3 -1 -1 -3 1 -2 3 2 2 3 2 3 -2 -1 -3 2 2 3 2 1 3 -2 -1 -1 3 2 3 -1 -2 1 2 1 -3 -3 2 2 -3 -1 2 3 1 1 -3 1 -2 -1 -2 -1
2 3 2 2 1 -2 -1 -2 3 -2 -2 -3 -2 -2 1 -2 -3 -1 3 -2 -2 -3 -2 -3 -2 1 1 -3 -2 1 1 3 -2 3 3 -1 -2 3 1 2 2 1 -2 -2 1 -3 -2 -1 -3 -1 -2
-2 -1 3 1 1 -3 1 -3 -1 -2 3 1 -3 1 2 -3 -1 -1 -1 -2 -1 2 -3 -2 -2 -1 -3 -3 -2 3 2 3 3 3 - 1 -2 -2 -1 -3 1 -3 -1 -2 -3 1 -3 -3 2 -3 -3
2 -1 -1 -2 -3 2 3 -2 -3 1 1 -3 -1 -2 -2 -3 -3 2 1 3 2 2 2 1 3 -1 3 2 1 1 1 2 3 1 2 -1 -3 1 3 -1 -2 -2 -3 -2 -2 1 2 - 1 -3 -3 -1 3 -1
-3 1 -3 -1 3 1 3 -1 3 1 3 2 1 -2 1 -2 -3 -2 -3 1 -2 -2 3 1 1 -3 -3 1 -3 -1 -2 -2 1 -2 -2 -2 -3 -2 -3 -3 -1 -3 2 2 3 -1 -2 3 -2 -1 -2
-1 -3 -2 -1 -2 -3 -1 -3 -2 -3 -2 3 -2 -1 -2 -1 3 -2 1 1 -2 3 -2 -2 -3 2 3 -2 1 3 1 2 3 1 -3 2 3 -2 1 2 -3 -1 3 -2 -3 -2 -3 1 2 3 -2
-3 -1 -2 -2 1 3 -2 3 1 2 -1 -1 -2 1 3 - 1 -2 -3 -2 1 -2 -1 3 -2 -1 3 -1 -1 3 2 -1 -1 -1 -2 -3 -2 -2 1 2 3 1 -2 -3 -2 -2 -3 -1 -3 1 -2
3 2 2 -3 1 -3 -3 -3 -1 3 1 3 3 -1 -1 -2 1 2 1 -3 -3 -3 -1 -1 -2 -1 -1 2 1 2 2 2 2 3 -1 -3 1 3 -2 -2 -1 -3 -3 1 3 -2 3 1 -2 1 3 -1 -2
-2 -2 1 3 -1 -1 -3 2 1 2 -3 -2 -3 -2 1 3 -2 -3 1 3 2 1 -2 1 3 3 -1 -1 2 2 -3 2 3 1 2 1 2 1 -3 2 1 -3 1 -2 1 -3 -1 -1 3 -1 -1 2 1 -3
-3 2 1 3 -2 -2 1 3 1 1 2 2 -3 1 2 1 -3 -2 1 3 1 2 3 3 2 -3 1 1 -2 3 -1 -1 2 2 1 3 -1 -3 1 2 -3 -1 -1 -3 -1 -3 -2 3 3 1 3 -2 -1 -2 3
3 -1 2 2 2 3 2 -1 -2 -1 -3 2 3 3 -2 -1 3 2 -3 2 1 2 2 -1 3 -2 -1 -3 2 -1 -2 -1 -3 2 1 1 2 1 3 -2 -1 2 3 -1 2 1 -2 -1 3 3 1 -3 -3 -2
-2 3 2 3 2 3 2 1 -2 -1 -1 2 2 -3 1 -3 2 2 -1 3 -2 1 1 3 -1 3 3 3 1 3 -1 -1 -2 1 3 2 2 -1 -3 2 1 - 3 -3 -2 1 1 2 1 -3 2 2 -3 -2 3 -1
-3 -2 3 3 -1 3 3 -2 -2 3 3 -1 -3 2 2 -1 -3 -1 -1 -3 -1 -3 -2 -1 -2 -1 -1 -2 3 1 -2 -1 -2 1 3 3 -1 3 2 -3 -3 -2 -2 3 1 1 2 2 1 -2 1
-2 -2 -2 -1 3 -1 -3 -3 -1 3 1 -3 1 3 1 3 -2 1 3 -2 -2 -3 2 1 -2 3 3 -1 - 1 -3 2 2 2 3 2 1 -2 -1 -1 3 3 1 2 3 -2 -2 -1 -3 1 3 1 -3 -3
-1 -1 -2 -1 2 3 3 -2 -2 3 1 1 3 -2 1 2 2 -1 -2 1 3 2 1 -3 -3 2 3 -1 3 2 -1 2 3 -2 -2 -2 -1 -2 -1 2 1 -2 3 -2 1 -3 -3 -3 2 -3 -3 -3
-1 -2 -1 -2 -1 2 -1 2 2 -1 -1 2 -3 -2 -3 -1 -3 2 -3 1 1 -3 -2 3 1 -3 -2 1 2 3 1 -2 -2 3 -1 2 3 1 2 -1 3 3 3 -2 -2 -1 -3 -1 3 2 1 -3
2 -3 2 -3 1 2 3 3 2 -3 2 1 -3 -2 -3 1 1 -3 -2 -2 1 1 2 -3 -1 -1 -2 -1 2 -1 -3 1 3 3 3 2 2 1 3 -1 -1 -2 -1 -2 -2 -3 -1 3 1 -2 1 -3 1
-3 1 3 3 1 3 2 3 -2 -3 2 -1 2 -1 3 -2 -2 -3 2 1 -3 2 -3 -1 -2 3 1 -3 -3 -1 2 1 -2 1 2 2 -3 2 1 1 -2 3 -2 -1 3 2 3 -1 2 -3 -3 -2
-1 -3 -2 1 1 -3 2 1 2 1 -3 -3 2 1 -2 -1 -3 -2 1 1 2 1 2 2 -3 2 -1 -2 -3 -2 1 3 3 3 3 -1 -2 3 -2 -2 -1 3 3 1 -2 1 2 2 -1 -3 2 2 2 -1

2 -1 -2 3 1 1 3 -2 -3 -3 -3 -2 -1 -1 -1 -2 3 3 2 3 1 2 -1 2 3 2 2 3 3 2 1 1 3 1 2 -3 1 2 3 -2 1 -2 3 -2 3 3 3 3 3 -1 2 1 3 1 1 3 -2
-2 -1 -2 -2 1 1 -2 -3 1 2 2 -1 -3 -3 -1 -3 2 1 -2 1 -3 -2 1 -3 2 -1 -1 -1 2 1 -2 -2 1 -3 -1 -2 -2 -1 2 2 -1 3 -2 1 -3 1 2 2 -3 -2 3
-2 -2 1 1 2 1 1 -2 -1 2 -3 -1 -3 -2 3 -1 -1 -3 2 3 1 1 2 1 -3 -2 -1 -3 -3 -2 -1 -1 2 2 3 -1 3 3 -2 -3 1 1 -2 -3 -3 -3 -1 -1 2 -1 -2
-2 - 2 1 -3 -1 2 1 2 3 1 3 -1 2 -1 -3 1 1 1 3 1 3 1 -2 1 -2 -1 -2 -2 -1 -1 -1 -3 -1 2 -1 2 -1 2 1 -2 -3 -1 -1 -1 -1 3 -2 3 -1 -1 -2
-2 -1 3 -1 -2 -3 -2 -3 2 2 3 2 1 -2 1 3 -1 -2 -3 -2 -3 -3 -3 -3 -2 -2 -2 -1 3 -2 -1 -3 -3 -1 -3 -1 -2 3 -2 -2 -3 -1 3 2 1 2 2 1 -3 -1 -3
-1 -3 -1 3 3 3 -2 -2 1 -2 -3 2 2 -1 3 -2 1 2 -3 -2 -1 -1 -3 -2 1 -3 2 1 3 -2 3 1 2 -1 3 -2 -2 1 2 3 2 -1 -2 -3 1 2 2 -1 3 3 -1 3 2 2
1 1 -3 1 3 3 3 -2 -2 -3 -2 3 -2 -1 2 2 2 2 3 3 -1 2 3 3 3 2 3 -1 -2 3 2 -1 3 -2 3 3 -2 -2 -1 -3 2 3 3 -2 -2 -1 -1 2 1 -2 -3 1 3 3 2
-1 -3 -1 -3 1 2 -1 -3 -2 3 -2 1 -3 1 -3 -2 1 -3 2 2 2 1 -3 -1 -1 2 -1 -3 -1 2 1 1 2 -1 3 -2 -3 1 2 2 1 -3 -1 -2 -3 -3 -1 3 1 2 3 1
-2 -1 -2 1 -3 1 -3 -3 -3 -2 1 -3 -3 -2 -1 3 1 3 1 -2 3 1 -2 -2 -2 1 1 3 -1 -3 2 -3 -2 1 -3 -2 1 1 -2 3 3 3 1 2 1 -2 1 -3 -2 -1 2 -3
-3 1 2 2 1 -3 -3 2 -3 -3 1 1 2 2 1 2 -3 2 -1 -1 -1 -2 3 -1 -2 -2 -2 3 -1 2 1 2 3 -2 3 3 2 - 1 2 3 2 3 -1 2 2 2 3 1 3 1 -2 3 1 1 3 3
-1 3 -1 -2 1 3 2 3 3 2 1 -2 -1 3 -1 -2 1 -2 1 -2 -2 3.

C Collisions for Morgenstern hashes, $q = 2$ and $\deg p(x) = 20$

Now we give a small example for our collision-finding algorithm. The polynomial we choose to target is $p(x) = x^{20}+x^{17}+x^{14}+x^{13}+x^{12}+x^{11}+x^9+x^7+x^5+x^3+x^2+x+1$. We choose $R = 10$ and generate random m and b''. After 3 random trials we get $m = x^9 + x^8 + x^7 + x^6 + x^5 + x^4$, $b'' = x^{10} + x^8 + x^5 + x^2 + 1$ so $k = 52$, $a = x^{52} + x^{48} + x^{36} + x^{32} + x^{30} + x^{25} + x^{24} + x^{22} + x^{20} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^9 + x^4 + x^3 + x + 1$, $b = x^{51} + x^{50} + x^{48} + x^{47} + x^{46} + x^{45} + x^{44} + x^{40} + x^{39} + x^{38} + x^{37} + x^{36} + x^{35} + x^{34} + x^{32} + x^{31} + x^{30} + x^{29} + x^{28} + x^{27} + x^{25} + x^{24} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{11} + x^{10} + x^9 + x^8 + x^5 + x^4 + x^3 + x^2 + x$ and $n = x^{62}+x^{61}+x^{59}+x^{57}+x^{55}+x^{53}+x^{52}+x^{51}+x^{50}+x^{49}+x^{48}+x^{46}+x^{45}+x^{40}+x^{37}+x^{31}+x^{29}+x^{28}+x^{26}+x^{25}+x^{24}+x^{23}+x^{16}+x^{15}+x^{13}+x^{12}+x^{10}+x^6+x^5+x^3+1$.

The polynomial n has three factors $n_1 = x^{56}+x^{54}+x^{53}+x^{50}+x^{48}+x^{46}+x^{44}+x^{40}+x^{36}+x^{34}+x^{33}+x^{30}+x^{29}+x^{22}+x^{20}+x^{18}+x^{13}+x^{11}+x^7+x^6+x^5+x^3+1$, $n_2 = x^4+x^3+x^2+x+1$ and $n_3 = x^2+x+1$ which are all of even degrees. For each factor n_i we compute α such that $\alpha^2+\alpha+1 \equiv 0 \bmod n_i$ and use this value and the continued fraction algorithm to recover (c_i, d_i) such that $c_i^2 + d_i^2 + c_i d_i \equiv 0 \bmod n_i$: we get $(c_1, d_1) = (x^{26} + x^{25} + x^{24} + x^{21} + x^{20} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^8 + x^6 + x^5 + x + 1, x^{28} + x^{23} + x^{21} + x^{19} + x^{15} + x^{13} + x^{10} + x^7 + x^5 + x^4 + x^2 + x + 1)$, $(c_2, d_2) = (x, x^2 + 1)$ and $(c_3, d_3) = (x, 1)$.

Combining these partial results we get $c = x^{51}+x^{50}+x^{47}+x^{41}+x^{40}+x^{36}+x^{31}+x^{27}+x^{26}+x^{25}+x^{24}+x^{23}+x^{22}+x^{20}+x^{18}+x^{17}+x^{16}+x^{14}+x^{12}+x^{11}+x^{10}+x^9+x^7+x^4$ and $d = x^{51} + x^{50} + x^{49} + x^{48} + x^{47} + x^{45} + x^{44} + x^{43} + x^{42} + x^{39} + x^{36} + x^{33} + x^{31} + x^{30} + x^{29} + x^{27} + x^{26} + x^{25} + x^{22} + x^{21} + x^{20} + x^{18} + x^{17} + x^{16} + x^{14} + x^{13} + x^9 + x^7 + 1$.

We can verify that

$$(a^2 + b^2 + ab) + (c^2 + d^2 + cd)x = (1 + x)^{2k}$$

and $(a, b, c, d) \equiv (1 + x)^k(1, 0, 0, 0) \bmod p$. We factorize the lifted matrix and, using the indices of the generators given in Section 5, we get the following collision with the void message: 0 2 0 2 0 1 2 1 2 1 2 1 2 0 2 0 2 0 2 0 2 0 1 0 2 0 2 1 0 2 1 0 1 0 2 1 0 1 2 1 2 1 2 1 0 2 1 0 1 2 0 1 0 1 0 1 0 2 1 0 1 2 0 2 1 2 0 2 0 1 2 0 2 0 1 0 2 1 2 1 0 2 0 1 0 1 2 0 2 0 2 0 2 0 1 2 1 0 2 0 2 1 0 1.

D Collisions for Morgenstern hashes, $q = 2$ and $\deg p(x) = 1024$

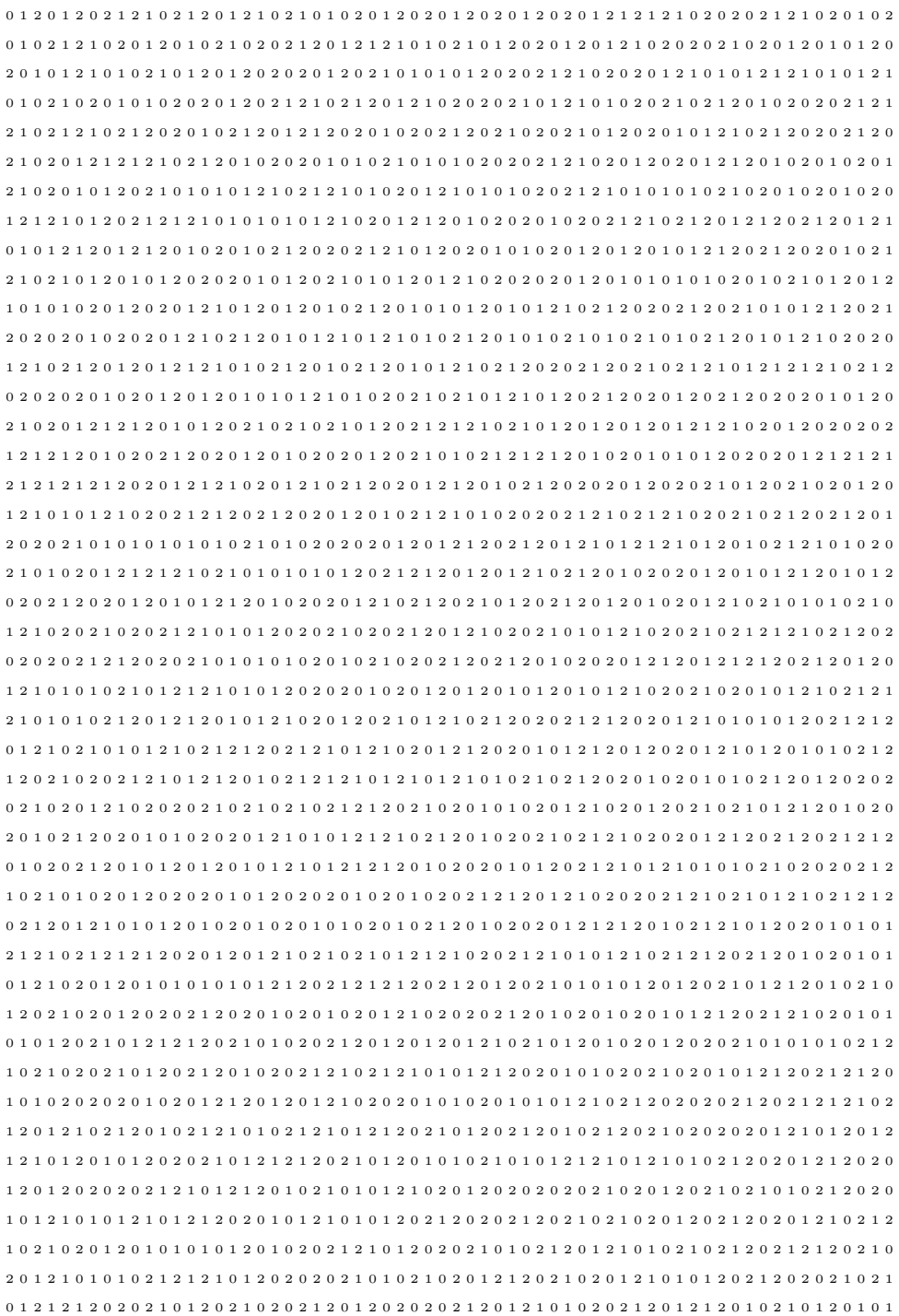
Let $p(x) = x^{1024} + x^{1023} + x^{1022} + x^{1020} + x^{1014} + x^{1013} + x^{1009} + x^{1006} + x^{1003} + x^{999} + x^{993} + x^{992} + x^{990} + x^{989} + x^{988} + x^{987} + x^{986} + x^{983} + x^{982} + x^{981} + x^{980} + x^{979} + x^{977} + x^{976} + x^{971} + x^{967} + x^{965} + x^{961} + x^{960} + x^{957} + x^{955} + x^{953} + x^{946} + x^{945} + x^{943} + x^{941} + x^{937} + x^{936} + x^{935} + x^{934} + x^{930} + x^{925} + x^{923} + x^{920} + x^{919} + x^{918} + x^{917} + x^{915} + x^{914} + x^{911} + x^{910} + x^{909} + x^{908} + x^{906} + x^{904} + x^{901} + x^{900} + x^{899} + x^{898} + x^{896} + x^{895} + x^{894} + x^{889} + x^{888} + x^{885} + x^{884} + x^{882} + x^{878} + x^{876} + x^{875} + x^{872} + x^{870} + x^{866} + x^{864} + x^{863} + x^{859} + x^{857} + x^{856} + x^{855} + x^{854} + x^{851} + x^{850} + x^{849} + x^{846} + x^{838} + x^{837} + x^{834} + x^{831} + x^{830} + x^{829} + x^{828} + x^{827} + x^{821} + x^{818} + x^{813} + x^{812} + x^{810} + x^{809} + x^{808} + x^{807} + x^{806} + x^{805} + x^{804} + x^{803} + x^{802} + x^{800} + x^{799} + x^{798} + x^{796} + x^{795} + x^{793} + x^{791} + x^{788} + x^{785} + x^{784} + x^{783} + x^{781} + x^{776} + x^{775} + x^{773} + x^{771} + x^{770} + x^{769} + x^{768} + x^{766} + x^{760} + x^{753} + x^{751} + x^{749} + x^{747} + x^{745} + x^{743} + x^{742} + x^{735} + x^{734} + x^{733} + x^{732} + x^{730} + x^{729} + x^{726} + x^{724} + x^{722} + x^{719} + x^{718} + x^{716} + x^{715} + x^{712} + x^{711} + x^{707} + x^{706} + x^{705} + x^{700} + x^{696} + x^{695} + x^{693} + x^{692} + x^{690} + x^{685} + x^{681} + x^{676} + x^{675} + x^{674} + x^{673} + x^{671} + x^{670} + x^{669} + x^{664} + x^{662} + x^{661} + x^{658} + x^{656} + x^{654} + x^{652} + x^{651} + x^{650} + x^{649} + x^{648} + x^{646} + x^{645} + x^{643} + x^{641} + x^{640} + x^{639} + x^{637} + x^{635} + x^{634} + x^{633} + x^{632} + x^{631} + x^{629} + x^{628} + x^{626} + x^{624} + x^{623} + x^{621} + x^{619} + x^{615} + x^{612} + x^{611} + x^{605} + x^{604} + x^{603} + x^{600} + x^{598} + x^{596} + x^{594} + x^{590} + x^{588} + x^{586} + x^{585} + x^{582} + x^{579} + x^{577} + x^{571} + x^{570} + x^{564} + x^{562} + x^{561} + x^{559} + x^{558} + x^{557} + x^{556} + x^{550} + x^{549} + x^{545} + x^{544} + x^{541} + x^{540} + x^{538} + x^{537} + x^{535} + x^{534} + x^{528} + x^{526} + x^{525} + x^{524} + x^{520} + x^{519} + x^{518} + x^{516} + x^{515} + x^{513} + x^{512} + x^{510} + x^{509} + x^{507} + x^{503} + x^{498} + x^{496} + x^{495} + x^{492} + x^{491} + x^{490} + x^{489} + x^{484} + x^{483} + x^{481} + x^{480} + x^{478} + x^{477} + x^{476} + x^{475} + x^{474} + x^{473} + x^{468} + x^{467} + x^{465} + x^{464} + x^{463} + x^{459} + x^{457} + x^{456} + x^{455} + x^{454} + x^{449} + x^{447} + x^{444} + x^{443} + x^{442} + x^{438} + x^{435} + x^{434} + x^{431} + x^{429} + x^{427} + x^{425} + x^{424} + x^{415} + x^{412} + x^{411} + x^{409} + x^{406} + x^{404} + x^{403} + x^{402} + x^{399} + x^{398} + x^{394} + x^{393} + x^{392} + x^{390} + x^{389} + x^{387} + x^{386} + x^{385} + x^{384} + x^{382} + x^{381} + x^{380} + x^{379} + x^{377} + x^{374} + x^{373} + x^{369} + x^{368} + x^{365} + x^{362} + x^{357} + x^{354} + x^{351} + x^{349} + x^{346} + x^{345} + x^{344} + x^{343} + x^{340} + x^{337} + x^{331} + x^{330} + x^{328} + x^{326} + x^{324} + x^{323} + x^{322} + x^{321} + x^{319} + x^{317} + x^{315} + x^{314} + x^{313} + x^{312} + x^{310} + x^{309} + x^{305} + x^{304} + x^{303} + x^{298} + x^{296} + x^{294} + x^{290} + x^{289} + x^{288} + x^{283} + x^{282} + x^{281} + x^{279} + x^{276} + x^{275} + x^{273} + x^{271} + x^{268} + x^{266} + x^{265} + x^{264} + x^{263} + x^{260} + x^{259} + x^{253} + x^{252} + x^{250} + x^{249} + x^{247} + x^{246} + x^{245} + x^{244} + x^{242} + x^{237} + x^{235} + x^{234} + x^{231} + x^{228} + x^{227} + x^{225} + x^{222} + x^{218} + x^{217} + x^{216} + x^{215} + x^{214} + x^{211} + x^{210} + x^{208} + x^{206} + x^{204} + x^{203} + x^{202} + x^{201} + x^{200} + x^{199} + x^{198} + x^{195} + x^{194} + x^{192} + x^{191} + x^{189} + x^{187} + x^{185} + x^{184} + x^{181} + x^{180} + x^{179} + x^{172} + x^{171} + x^{170} + x^{165} + x^{164} + x^{162} + x^{161} + x^{159} + x^{157} + x^{153} + x^{152} + x^{151} + x^{149} + x^{148} + x^{146} + x^{145} + x^{143} + x^{140} + x^{137} + x^{134} + x^{133} + x^{128} + x^{127} + x^{125} + x^{123} + x^{121} + x^{120} + x^{119} + x^{115} + x^{113} + x^{110} + x^{108} + x^{107} + x^{105} + x^{103} + x^{102} + x^{100} + x^{99} + x^{96} + x^{94} + x^{89} + x^{87} + x^{86} + x^{83} + x^{82} + x^{81} + x^{79} + x^{77} + x^{76} + x^{73} + x^{70} + x^{69} + x^{68} + x^{64} + x^{62} + x^{61} + x^{59} + x^{57} + x^{56} + x^{55} + x^{54} + x^{53} + x^{51} + x^{50} + x^{49} + x^{48} + x^{47} + x^{45} + x^{44} + x^{41} + x^{40} + x^{39} + x^{37} + x^{34} + x^{31} + x^{27} + x^{22} + x^{18} + x^{14} + x^{10} + x^9 + x^8 + x^5 + x^2 + x + 1.$

Then the following sequence collides with the void sequence:

0 2 0 1 0 2 0 2 0 1 2 1 0 1 0 2 1 0 2 0 1 2 0 1 0 2 1 0 2 0 1 0 2 1 0 1 2 0 2 0 2 0 2 1 0 2 0 1 2 1 2 0 1 2 0 2 0 1 2 1 0 2

1 0 1 2 1 2 0 2 0 2 1 2 1 2 0 1 2 0 2 0 1 2 0 1 2 1 2 1 2 1 0 2 1 0 1 2 0 1 0 1 0 2 0 2 1 2 0 2 0 1 2 0 1 0 2 1 2 0 2 1 0 2 0 2

0 1 2 0 1 0 2 1 0 2 0 1 2 0 2 1 2 1 2 1 2 0 2 0 1 2 1 2 0 2 1 0 2 1 0 1 0 1 2 0 1 0 1 0 2 0 2 0 2 0 1 0 2 0 2 1 0 2 1 2 0 1 0 2



0 1 0 2 0 2 0 1 2 1 0 1 2 0 1 0 1 2 0 2 0 1 2 1 0 1 2 0 1 2 1 2 1 2 0 2 0 2 0 2 1 0 2 0 1 2 0 2 0 2 1 0 1 0 2 0 1 2 1 2 1 0 2 0
2 0 2 1 0 2 1 2 0 2 0 2 1 2 1 2 1 0 2 0 2 1 2 0 1 2 0 2 0 1 2 1 2 0 2 0 2 0 1 2 0 2 1 2 1 0 1 0 2 0 1 0 1 0 2 1 0 1 2 1 2 1 0 1
0 1 0 1 0 1 2 0 1 2 0 2 1 0 1 2 0 2 1 2 1 0 2 0 1 2 1 0 2 1 0 2 0 1 0 1 0 2 0 2 0 1 0 1 0 1 0 1 2 1 2 0 1 2 0 1 0 1 2 0 2 0 2 0
1 2 1 2 0 1 2 1 0 1 0 1 0 2 1 0 2 1 0 2 0 2 0 2 1 0 1 0 1 2 1 2 0 2 0 1 0 1 0 1 2 0 1 0 1 0 2 1 2 0 2 1 2 0 2 0 2 1 0 1 0 1 0 1
2 0 1 2 0 2 0 1 2 0 2 0 1 2 1 2 0 2 0 2 1 0 1 0 1 2 1 0 2 1 2 0 2 0 2 1 2 0 1 2 0 2 0 2 0 2 1 0 2 1 2 1 0 2 0 1 2 0 2 0 1 0 2 1
0 1 0 1 2 0 1 0 2 0 1 2 0 1 2 0 1 2 1 2 1 0 1 0 2 1 0 1 2 0 2 0 1 0 1 0 1 0 1 0 2 1 0 2 1 0 1 0 2 0 2 0 1 2 0 2 0 2 0 2 1 2 0 2
0 1 0 2 0 1 2 1 2 1 2 0 2 1 2 1 2 0 1 2 0 1 0 2 0 2 1 2 1 0 2 0 2 0 2 0 2 0 1 2 1 0 2 1 0 1 2 1 0 1 0 1 0 2 0 1 0 1 2 0 2 1 2 1
2 0 2 1 0 2 0 1 0 1 0 1 2 0 1 2 1 2 0 1 2 0 2 1 2 1 0 1 0 2 0 2 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 0 1 0 1 0 2 1 0 1 2 0 1 2 1 2 1 2
0 1 0 1 0 2 1 2 0 1 0 1 0 2 0 1 0 2 0 2 1 2 1 2 0 2 0 2 0 2 1 0 1 2 1 2 0 1 0 1 2 1 0 2 0 2 0 2 1 0 1 2 1 0 2 1 0 2 1 0 2 0 2
1 2 1 2 0 2 0 2 0 2 1 2 0 1 0 2 1 2 0 2 1 2 0 1 0 2 1 2 0 1 2 1 0 2 0 2 1 2 0 2 1 0 1 2 1 0 2 0 1 0 1 0 2 1 2 0 1 0 1 0 1 2 0 2
1 2 0 1 2 0 1 2 0 1 0 2 1 2 1 0 1 2 1 2 1 2 0 2 0 1 2 0 1 2 0 2 0 2 1 0 1 0 2 0 1 0 2 0 1 2 1 0 1 2 1 0 2 1 2 1 0 2 0 1 0 1 2 0
1 0 1 2 1 0 2 1 2 1 0 2 0 1 2 1 0 2 0 1 0 2 0 2 0 1 0 2 0 1 2 0 1 2 1 2 0 1 0 1 2 1 2 0 1 0 2 1 2 1 2 1 2 1 2 0 2 1 0 1 2 1 2 0
1 0 2 0 1 0 1 2 1 2 0 1 2 1 0 2 1 0 1 2 0 1 2 0 1 0 2 1 2 1 2 1 0 2 0 1 2 0 2 1 0 2 1 2 1 2 0 1 0 2 1 0 2 1 2 0 1 0 2 1 0 2 1 2 0 1 2
1 2 1 0 1 0 2 1 0 2 0 1 2 1 2 0 2 1 2 1 2 0 2 0 2 1 0 2 1 0 2 0 2 0 1 2 1 0 2 0 1 2 0 1 0 1 0 1 0 1 0 1 0 1 2 0 1 2 1 0 2 0 1 2
1 2 1 0 2 0 1 0 2 1 0 2 1 0 1 0 1 0 2 0 1 2 1 0 1 0 1 0 1 0 2 1 2 1 2 1 0 2 1 0 1 0 2 1 2 1 2 0 2 0 1 0 2 0 1 2 0 2 1 2 1 2
0 1 2 0 2 1 0 1 2 1 0 2 0 2 1 2 0 1 2 1 0 2 1 0 1 0 2 1 2 1 2 0 2 1 0 2 0 2 1 2 1 0 1 0 2 0 1 2 1 2 1 2 1 0 2 1 2 0 2 1 0 1 2 0
1 2 1 0 2 0 1 2 1 2 0 1 2 0 2 0 1 0 2 0 2 1 0 1 2 0 1 2 1 0 2 0 2 1 2 0 2 0 1 2 1 0 2 0 1 2 0 1 0 1 0 1 0 1 0 1 2 0 1 2 1 0 2 0 1 2 .