Public-key encryption from a trapdoor one-way embedding of $SL_2(\mathbb{N})$

Robert Hines
alphanumericnonsense@gmail.com

September 10, 2024

Abstract

We obfuscate words of a given length in a free monoid on two generators with a simple factorization algorithm (namely $SL_2(\mathbb{N})$) to create a public-key encryption scheme. We provide a reference implementation in Python and suggested parameters. The security analysis is between weak and non-existent, left to future work.

1 Introduction

The idea for the encryption scheme is straightforward. First map bitstrings of length λ to products of length λ in a free monoid on two generators. The particular monoid instance should have a simple factorization algorithm. Next, homomorphically obfuscate the monoid (trapdoor one-way embedding) and provide the obfuscated generators as a public key. We provide a detailed algorithmic instantiation of this idea below with the monoid $SL_2(\mathbb{N})$ and a Python implementation of the resulting scheme for experimentation.

While there is much to be done concerning cryptanalysis, we give a list of important questions affecting security and practicality of the scheme.

2 Prior art

An existing "non-commutative knapsack" approach overlapping conceptually with our proposal is [1]. Our use of the algebraic structure of $SL_2(\mathbb{N})$ (as opposed to trial-and-error and heuristics) and smaller public keys (two generators instead of a knapsack setup) is arguably more elegant if nothing else.

3 The medium: $SL_2(\mathbb{N})$

Let $L = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $R = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. These two matrices generate $SL_2(\mathbb{N})$ as free monoid on two generators, i.e. every 2×2 matrix with non-negative integer entries and determinant one can be written uniquely as a finite word in the alphabet $\{L, R\}$. We note for later use that a product/word of length k in $\{L, R\}$ has non-negative integer entries bounded by $m = 2^k$. This bound could be made smaller, e.g. the kth Fibonacci number F_k is the maximum matrix entry of words of length k, but the bound $m = 2^k$ will suffice for our purposes.

Moreover, given an element $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{N})$, we can recover the factorization using some variant of the Euclidean algorithm or continued fractions. For instance:

- If $a/c \leq 1$ then $M = LM'$, and if $a/c > 1$ (including $1/0 = \infty$), then $M = RM'$.
- Left multiply by L^{-1} or R^{-1} and continue until $M' = 1$.

4 The trapdoor: modular reduction, conjugation, increasing dimension, and random generators.

We start with a naive attempt at a trapdoor and describe refinements to obtain something useful (or so we hope).

4.1 First attempt (conjugate)

Denote by $SL_2^{(\lambda)}(\mathbb{N})$ the set of all words of length λ in the alphabet $\{L, R\}$. As a first attempt to obfuscate $M \in SL_2^{\lambda}(\mathbb{N})$, we work modulo m, where m is any integer greater than $\max\{\|A\|_\infty : A \in SL_2^{(\lambda)}(\mathbb{N})\}$, and conjugate by a uniformly random element $S \in GL_2(\mathbb{Z}/m\mathbb{Z})$, $M \mapsto S^{-1}MS$.

However, $\{S^{-1}LS, S^{-1}RS\}$ leaks too much information, namely the algebraic relations

$$S^{-1}LS = \frac{1}{ad-bc}\begin{pmatrix} -ab+ad-bc & -b^2 \\ a^2 & ab+ad-bc \end{pmatrix},$$

and

$$S^{-1}RS = \frac{1}{ad-bc}\begin{pmatrix} ad-bc+cd & d^2 \\ -c^2 & ad-bc-cd \end{pmatrix},$$

where $S = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. These are enough to recover the secret S by finding square roots modulo m.

4.2 Strike two (increase the dimension)

We can first increase the dimension, setting a parameter $n \geq 2$, embedding

$$L \mapsto \widetilde{L} = \begin{pmatrix} I_n & 0 \\ I_n & I_n \end{pmatrix}, \quad R \mapsto \widetilde{R} = \begin{pmatrix} I_n & I_n \\ 0 & I_n \end{pmatrix},$$

and conjugating by a secret $S \in GL_{2n}(\mathbb{Z}/m\mathbb{Z})$. By taking $n > 1$, we get equations similar to the first attempt,

$$S^{-1}\widetilde{L}S = \begin{pmatrix} EA+FA+FC & EB+FB+FD \\ GA+HA+HC & GB+HB+HD \end{pmatrix},$$

and

$$S^{-1}\widetilde{R}S = \begin{pmatrix} EA+EC+FC & EB+ED+FD \\ GA+GC+HC & GB+GD+HD \end{pmatrix},$$

where $S = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, $S^{-1} = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$. One would hope that these are more difficult to solve, but perhaps cryptographically secure values of n would make public key and ciphertext sizes impractical.

4.3 Third time's a charm (use random generators)

Instead of using the generators $\{L, R\}$, we could randomly choose $\{G_0, G_1\}$, each a word of length l in $\{L, R\}$, i.e. $\{G_0, G_1\} \subseteq SL_2^{(l)}(\mathbb{N})$. For large enough l, the resulting public key and plain/cipher pairs should be secure. The cost to pay for this is an increase in the bound m, namely we must work with $SL_2^{(l\lambda)}(\mathbb{N})$ and take something like $m = 2^{l\lambda}$. For practicality, l should be on the small side.

4.4 Four (all of the above)?

Finally, we could combine the second and third attempts by using random generators and larger matrices. Whether or not this is prudent depends on a more careful analysis of the weaknesses of each.

5 Public key encryption scheme

In the following, we choose parameters λ, l, and m, and let G_0, G_1 be obtained from randomly selected words of length l in $\{L, R\}$.

5.1 Key generation

The secret key consists of S, a uniformly random element of $GL_{2n}(\mathbb{Z}/m\mathbb{Z})$, and the random matrices $\{G_0, G_1\}$. The public key is the pair $P_0 = S^{-1}\widetilde{G_0}S$, $P_1 = S^{-1}\widetilde{G_1}S$, with all arithmetic operations over $\mathbb{Z}/m\mathbb{Z}$.

5.2 Encryption

Let μ be a λ-bit message. To encrypt $\mu = \mu_0\mu_1\cdots\mu_{\lambda-1}$, form the product $C = \prod_{i=0}^{\lambda-1} P_{\mu_i} \in SL_{2n}(\mathbb{Z}/m\mathbb{Z})$, with all arithmetic modulo m.

5.3 Decryption

To decrypt the ciphertext C, first conjugate by S^{-1} to get $SCS^{-1} = \prod_{i=0}^{\lambda-1} \widetilde{G_{\mu_i}}$ (arithmetic modulo m). This is now an unobfuscated element of $SL_2(\mathbb{N})$ (after reducing a block "constant" $2n \times 2n$ matrix to a 2×2 matrix). This can be factored easily as described in an earlier section to recover μ.

6 Security and parameter selection

Let $F_2^{(\lambda)}$ be the set of words of length λ in the free monoid on two generators. Is the embedding defined by the public key, $\epsilon_{P_0,P_1} : F_2^{(\lambda)} \hookrightarrow SL_{2n}(\mathbb{Z}/m\mathbb{Z})$, one-way? While this is the paramount security question, we leave it aside for the moment and consider potential practicality first. We have

$$|\mathsf{sk}| = 4n^2l\lambda + 2l, \quad |\mathsf{pk}| = 8n^2l\lambda, \quad |\mathsf{ct}| = 4n^2l\lambda.$$

Another concern is the large $l\lambda$-bit integer arithmetic. As a starting point, we could take $l = \lambda = 256$ and $n = 1$, naive 256-bit security against finding the generators or plaintext recovery with 256-bit plaintexts. While key and ciphertext sizes are "reasonable," the 2^{16}-bit multiplication is not. Sliding the parameters to $n = 16$, $l = 1$, $\lambda = 256$ keeps the same key and ciphertext sizes and reasonable 256-bit arithmetic. A parameter choice in between the two extremes is $l = 16$, $n = 4$, $\lambda = 256$ (again keeping key and ciphertext sizes fixed), with 4096-bit multiplication (in line with modern RSA). At this point considering smaller bounds on m starts looking good, with smaller exponential growth in m as a function of λ.

6.1 Attacking conjugation

Suppose $n = 1$ for simplicity. The public key (P_0, P_1) and ciphertext C don't hide the conjugacy classes of (G_0, G_1) or μ encoded as a product of $\{L, R\}$ matrices, i.e. the trace is preserved or almost preserved modulo m. The 2×2 matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ we're looking for (say one of G_0, G_1, or μ) is a solution to

$$\begin{cases} ad - bc = 1, \\ a + d = t \quad (t \text{ is the trace}), \\ 0 \le a, b, c, d < m = 2^k, \text{ e.g. } 2^l \text{ or } 2^{l\lambda}, \\ \mathsf{len}_{\{L,R\}}(M) = k, \text{ e.g. } l \text{ or } l\lambda, \\ M' := S^{-1}MS \bmod m = 2^k \text{ is random and known}, \end{cases}$$

where t is the known trace. To the best of our knowledge, these are essentially the constraints, although some number theory may help (M is probably hyperbolic, stabilizes its fixed point form $\pm F(x, y) = cx^2 + (d - a)xy - by^2$ of discriminant $D = t^2 - 4$, the class number of $\mathbb{Q}(\sqrt{D})$ is probably 1, Pell's equation is easy to solve via continued fractions, etc.).

In other words the first problem is:

Given a random element $G = S^{-1}MS$ (S uniformly random on $GL_2(\mathbb{Z}/m\mathbb{Z})$, $m = 2^k$) of the conjugacy class of an element $M \in SL_2^{(k)}(\mathbb{N})$ (i.e. length k in $\{L, R\}$), what is the complexity of determining M as a function of k?

Solving the problem above with $k = l\lambda$ suffices for plaintext recovery. It is also a first step towards secret recovery, i.e. finding G_0 and/or G_1 from $P_0 = S^{-1}G_0S$, $P_1 = S^{-1}G_1S$ before solving for the secret S.

Another question regarding the conjugacy class search is:

What is the distribution of $\mathsf{tr}(M)$ for $M \in SL_2^{(k)}(\mathbb{N})$?

For instance, if this isn't uniform enough on its support (i.e. traces are biased on length k words), that could weaken the cryptosystem if exploitable. There are obvious non-uniformities, e.g. L^k and R^k are the only parabolics (trace 2).

The trace distribution may be explicitly computable; there's a sort of "non-commutative" recurrence in the Cayley tree of the monoid with respect to the generators $\{L, R\}$, namely

$$\mathsf{tr}(ML) = \mathsf{tr}(M) + b, \quad \mathsf{tr}(MR) = \mathsf{tr}(M) + c$$

where $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Experimentally, Figure 1 shows distributions of traces for words of length $k = 22$, 23, and 24.

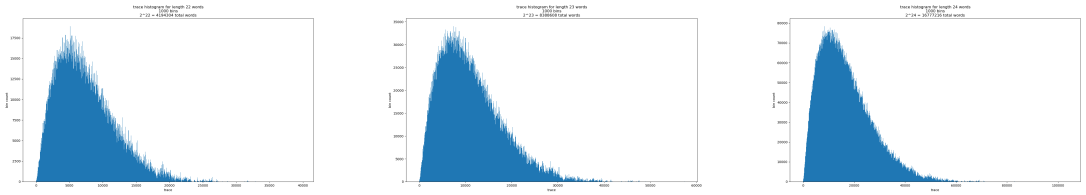

Figure 1: Trace distributions for words of length 22, 23, and 24. Histograms with 1000 bins of equal size.

6.2 Attacking the secret key S

Suppose we have known or chosen plain or ciphertext, or better yet the generators G_0, G_1 (which is the case with $l = 1$). When $n = 1$, we've already seen the resulting system of equations is easy to solve, taking square roots modulo m. Does increasing n make the problem more difficult?

What is the computational complexity of finding S from known M and $S^{-1}\widetilde{M}S$ as a function of n?

Using the same notation as before ($n \times n$ blocks for S, S^{-1}),

$$S = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad S^{-1} = \begin{pmatrix} E & F \\ G & H \end{pmatrix}, \quad M = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

we have

$$S^{-1}\widetilde{M}S = \begin{pmatrix} aEA + cFA + bEC + dFC & aEB + cFB + bED + dFD \\ aGA + cHA + bGC + dHC & aGB + cHB + bGD + dHD \end{pmatrix}.$$

With minor assumptions, we can write expressions for S^{-1} in terms of S (cf. [2]), e.g. if A is invertible then $(D - CA^{-1}B)$ is invertible and

$$S^{-1} = \begin{pmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{pmatrix}.$$

For simplicity, say $a = 1$, $b = c = d = 0$, so that

$$S^{-1}\widetilde{[1, 0; 0, 0]}S = \begin{pmatrix} I + A^{-1}B(D - CA^{-1}B)^{-1}C & A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}B \\ (D - CA^{-1}B)^{-1}C & (D - CA^{-1}B)^{-1}CA^{-1}B \end{pmatrix}.$$

This still seems formidable, with $\approx 4n^2$ independent variables and $4n^2$ equations, with a solution determined up to the centralizer of M.

7 Implementation, example, challenge

7.1 Implementation

A Python implementation is provided (`sl2_pke.py`[1]), tested at the first and third parameter sets from the previous section. The implementation is not optimized so takes a while for the suggested parameters. (It seems large n is a computational barrier for our implementation.) The times listed include key generation, encryption, and decryption on a typical desktop computer (function `test` in `sl2_pke.py`).

1. $l = \lambda = 256$, $n = 1$: 19.68 seconds (average over 100 trials).
2. $l = 1$, $\lambda = 256$, $n = 16$: ∞ (i.e. more than I wanted to wait).
3. $l = 16$, $\lambda = 256$, $n = 4$: 14.26 seconds (average over 100 trials).

In the implementation, the secret key consists of binary $\{L, R\}$ representations of G_0, G_1, along with S, S^{-1} so as to avoid additional computation during decryption. The public key is just (P_0, P_1). For all three parameter sets we have

$$|\mathsf{pk}| = 8n^2 l\lambda = 2^{19} \text{ bits}, \quad |\mathsf{ct}| = 4n^2 l\lambda = 2^{18} \text{ bits}.$$

The bulk of the time is spent in key generation, computing determinants and inverses (rejection sampling that $\det(S)$ is a unit and S is invertible). The implementation computes determinants recursively using expansion along the first row, and computes inverses via the adjugate. One could work with m prime to increase the proportion of invertible S and reduce the number of iterations in the rejection sampling.

7.2 Example

For a small example, let's take $l = 8$, $\lambda = 16$, and $n = 2$.

```
>>> import sl2_pke
>>> sl2_pke.test(8,16,2,verbose=True)

parameters: l=8, lmbd=16, n=2
elapsed time: 0.003463268280029297 seconds

secret G0bin = [0, 1, 0, 1, 1, 1, 0, 1]
secret G1bin = [0, 1, 1, 0, 1, 1, 1, 0]
secret S =
[[180044238861648907639530005239728443402, 786719381105032563565359454594936892 60,
3282086030435186554538364578683823315 20, 189858830372659235000974137164677995131],
[155672343735660761206347097294905535103, 106372412637255346718676377918699425235,
906916511390813477748389652 8357027578, 227922651931749744248605342978520132122],
[309496835986684457405760293018676 3998, 266641831191564483984905159250413954559,
209985679579407539411100359980165 8139, 160767525724397655375861666686953163840],
[178211095150397608370330506792434650267, 180684221329008827603458918903876565245,
139118072370641183190339253932365801613, 216052035797160283495565699148439592169]]
secret Sinv =
[[307192635118851668906543466683872777855, 269289625030035436706852480597147233360,
159981812487302740355310113004898255387, 534081205346135889309477276487505106 91],
[112181663788938666557979307486052261417, 102697689286225385400371584388074308505,
322861037399008184822766002568529216261, 174054502788728153025050567413076609419],
[237808841455499161862946284947239379397, 223110780377640152845753329325645361211,
146944259742648249323012403503666467604, 294481618604031787995793308484031011747],
```

[1] https://github.com/alphanumericnonsense/sl2_pke

```
[251449403454450419481736276209944042317, 505032914541368514942863361694491761888,
306319459087838687149182537269619092194, 230711238969461627876606261311237484890]]

public P0 =
[[137413500478426757237729785498897169715, 231239968320834077426658982689680266534,
691464954804205362788755826151702107724, 331527085370184892153210770948939809461,
[330162081674891111389361987769479757090, 191423324484097502089583299931615008607,
128679066393417423397836870943530733124, 275709108234098948675338258762948236678],
[885632543994210306250996913658435252541, 772013150859512786874303628200534419981,
958812034979231666647519940077934026531, 278369838638176645352618961861731543340],
[368463957120985195191838476867234457401, 173455529802750487353281038241693342532,
244424815999490852163822120880693491112, 255846705381429500934684135425230842005]]
public P1 =
[[247715784565787532463649675348437760736, 590825466844556606389825857084248908631,
324033841401195836787414966110816031953, 521006666270398717523877609458360116641],
[253980257488570419290325329821150515583, 258476128188832362048216706929886240643,
267003341997740339329266016284380686648, 132860127655589835033045029152588691757],
[251124012141290027957796299197635681645, 221946805871930782704353653080782150876,
230415952009162124747908556224420704349, 308271111609761317369148020906687311082],
[103786741178691326012204819233272678118, 160618027061409062154728853652258911287,
299106533718846902032063526114038025197, 284239235999033371130348883792559928698]]

mu = 0xa7b4
C =
[[848468826334853463024156874635406713, 109823726553789133715145299719758574752,
195790665770780782515747872767967575476, 312932876026461235025803616950559157428],
[459707847752144452495528248046730304961, 138292064403972714254128143616197175537,
286470300973112865023394401799505845201, 528706732244182642075412833002883132721],
[218266962851451516080547949287673032976, 169447180539165412579546408985241709321,
628417923612149064742280626876285590931, 204026872879168090956118039963279146212],
[296989855556537546749115273042727692641, 551421844055170126380869452680619630201,
242681753657713470273331324236315756361, 543016275222668140893728382683394823411]]
mu_recovered = 0xa7b4

SUCCESS
```

7.3 Challenge

Data for the first and third parameter sets can easily be obtained from the given implementation.

1. $l = \lambda = 256$, $n = 1$,
2. $l = 1$, $\lambda = 256$, $n = 16$ (bad implementation or computationally infeasible),
3. $l = 16$, $\lambda = 256$, $n = 4$.

We welcome any attempt to break these (or other) parameter sets, or any further analysis of the scheme.

8 A final improvement (not implemented)

In light of some of the observed limitations, (large integer arithmetic, non-uniform trace distribution), and with an eye towards a key encapsulation mechanism, one potential improvement is to produce G_0, G_1, and μ with traces uniformly in the "bulk" of the trace distribution and use an appropriately smaller value of m. The price to pay is an increase in the size of l and λ.

For instance, if some large fraction f of the messages are in the intersection

$$\{M : t_0 \leq \mathsf{tr}(M) \leq t_1\} \cap \{M : \|M\|_\infty \leq m_{red}\},$$

say with traces in an interval around the trace t_{max} at which the peak density occurs and m_{red} around half that value, this should result in a small loss in the message space (i.e. a small increase in l, λ) with a much smaller reduction constant $m_{red} \ll m \approx 2^n$. In other words, the benefits could outweigh the cost to give a more practical scheme.

Of course, for suitable parameters to be chosen, the joint distribution of the traces and of $\|M\|_\infty$ needs more detailed analysis. For a small experiment, Figure 2 shows the L^∞ distribution for words of length 15, 20, and 25.

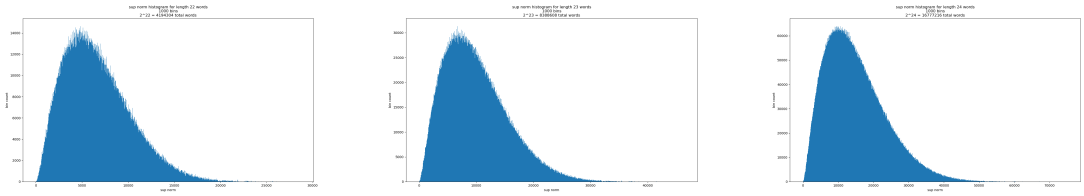

Figure 2: L^∞ distributions for words of length 22, 23, and 24. Histograms with 1000 bins of equal size.

Figure 3 gives 2D histograms of the joint distribution for lengths 22, 23, and 24. The obvious linear relationship seems to be $\mathsf{tr}(M) \approx \frac{9}{8}\|M\|_\infty$.

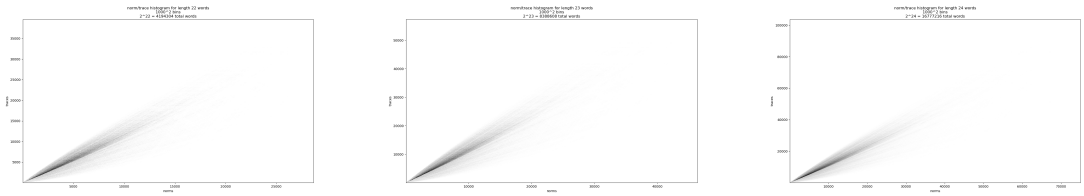

Figure 3: Joint distributions for words of length 22, 23, and 24. Histgrams with 1000 bins in each dimension.

To state the question more plainly for reference:

> What is the distribution of $\{(\mathsf{tr}(M), \|M\|_\infty) : M \in SL_2^{(k)}(\mathbb{N})\}$?

To repeat ourselves, a modest increase in k to uniformize the trace distribution and decrease the modular reduction bound m should increase both the security and practicality of the scheme.

9 Questions

We summarize some of the mathematical questions raised above, both for security and practicality of the scheme.

1. What is the distribution of $\{\mathsf{tr}(M) : M \in SL_2^{(k)}(\mathbb{N})\}$?
2. What is the distribution of $\{\|M\|_\infty : M \in SL_2^{(k)}(\mathbb{N})\}$?
3. What is the joint distribution of the two quantities above?
4. What is the difficulty (or even a more precise algebraic statement) of the secret key search problem, i.e. finding a uniformly random $S \in GL_2(\mathbb{Z}/m\mathbb{Z})$ given a known $M \in SL_2^{(k)}(\mathbb{N})$ and a known element $S^{-1}\widetilde{M}S$ of the conjugacy class of its "restriction of scalars" (trivially increasing the dimensions from 2×2 to $2n \times 2n$)?

Regarding the first three questions, there is the obvious binomial distribution on the "abelianized" structure, i.e. in terms of the number of instances of L and R in a given word, but the trace and L^∞ distributions depend on the non-commutativity, e.g. more "switching" activity increases the trace and maximum matrix entry. However, in lieu of more detailed information on the actual distributions, there may be useful bounds

on the trace and sup norm as a function of the relative frequency of L or R in a given word. Such bounds could also greatly reduce the computation needed for sampling of G_0, G_1, μ (i.e.sample based on proportions of L and R before matrix computations).

Towards investigating these questions, we note that the reference [3] shows that

$$\lim_{x\to\infty} \frac{1}{x} |\{n \leq x : \phi(n) \in (a,b]\}| = \int_a^b \delta(t)dt \quad (\text{for } n \geq 3)$$

for some smooth probability distribution δ, where

$$\phi(n) = \frac{\Phi(n)}{n \log n}, \quad \Phi(n) = |\{M \in SL_2(\mathbb{N}) \ : \ \mathsf{tr}(M) = n\}|,$$

and [4] gives an asymptotic for the summatory function

$$\Psi(N) = \sum_{3\leq n\leq N} \Phi(n) \sim \frac{6}{\pi^2} N^2 \log N$$

with further refinements in [5], [6].

10 Final thoughts on security

The scheme outlined above doesn't satisfy basic indistinguishability for two obvious reasons. One is the lack of randomness, which can easily be remedied, say by including a random mask ρ with the ciphertext and encrypting masked plaintext $\mu \oplus \rho$. The second is the trace distinguisher, with potential mitigations discussed above (restricting the range of allowed traces). For the purposes of a KEM, a good lower bound on the entropy loss due to knowledge of the trace should suffice.

One nice aspect of the decryption is that it can detect malformed ciphertext. If presented with ciphertext that is too short, too long, or built from the wrong generators, etc., the Euclidean algorithm will not reduce the ciphertext to the identity in the correct number of steps, and the decryption algorithm can respond with explicit or implicit rejection.

We also note that the ciphertext is malleable to some extent. Given valid ciphertext, one can attempt to remove and replace bits from the beginning and end of the ciphertext (probability 2^{-k} of guessing k bits to remove). This could be mitigated by random padding, i.e. only the "middle" of the plaintext is considered relevant.

We welcome any and all to attack and improve the scheme or further develop our admittedly weak security analysis.

References

[1] R. Geraud-Stewart and D. Naccache, "New public-key cryptosystem blueprints using matrix products in $\mathbb{F}_p$." Cryptology ePrint Archive, Paper 2023/1745, 2023.Available: https://eprint.iacr.org/2023/1745

[2] T.-T. Lu and S.-H. Shiou, "Inverses of 2 × 2 block matrices," *Computers & Mathematics with Applications*, vol. 43, no. 1, pp. 119–129, 2002, doi: https://doi.org/10.1016/S0898-1221(01)00278-4.

[3] M. Peter, "The limit distribution of a number theoretic function arising from a problem in statistical mechanics," *Journal of Number Theory*, vol. 90, no. 2, pp. 265–280, 2001, doi: https://doi.org/10.1006/jnth.2001.2666.

[4] J. Kallies, A. Özlük, M. Peter, and C. Snyder, "On asymptotic properties of a number theoretic function arising out of a spin chain model in statistical mechanics," *Communications in Mathematical Physics*, vol. 222, no. 1, pp. 9–43, Aug. 2001, doi: 10.1007/s002200100495.

[5] F. P. Boca, "Products of matrices and and the distribution of reduced quadratic irrationals," *Journal für die reine und angewandte Mathematik*, vol. 2007, no. 606, pp. 149–165, 2007, doi: doi:10.1515/CRELLE.2007.038.

[6] A. V. Ustinov, "Spin chains and arnold's problem on the gauss-kuz'min statistics for quadratic irrationals," *Sbornik: Mathematics*, vol. 204, no. 5, p. 762, Jan. 2013, doi: 10.1070/SM2013v204n05ABEH004319.