IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 3, 20 pages.

https://doi.org/10.62056/avrxrudhdj

Attacking trapdoors from matrix products

Thomas Decru[1], Tako Boris Fouotsa[2], Paul Frixons[1], Valerie Gilchrist[1] and Christophe Petit[1,3]

[1] Université Libre de Bruxelles, Brussels, Belgium
[2] EPFL, Lausanne, Switzerland
[3] University of Birmingham, Birmingham, England

Abstract. Recently, Geraud-Stewart and Naccache proposed two trapdoors based on matrix products. In this paper, we answer the call for cryptanalysis. We explore how using the trace and determinant of a matrix can be used to attack their constructions. We fully break their first construction in a polynomial-time attack. We show an information leak in the second construction using characteristic polynomials, and provide two attacks that decrease the bit security by about half.

1 Introduction

In 2023, Geraud-Stewart and Naccache proposed a new trapdoor based on matrix products [GSN23]. From a set of public invertible matrices $A_1, ..., A_k$, it sends the permutation σ to the product $\prod_{i=1}^{k} A_{\sigma(i)}$. Even if the practical encryption does not become competitive to already known procedures, the prospect of a new family of trapdoors is enriching for cryptography in general as it could lead to new applications. The simplicity of the scheme and the concepts are also attractive in itself when thinking about widespread implementations.

That being said, trust can only be given to a scheme once it has received careful inspection and resisted the various attempts of breaking from the cryptographic community. In the last year, no work assessing the security of these schemes has been released. This paper is inscribed in this procedure, and in fact shows that the constructions, as they are, do not meet their claimed security levels.

Related work. These trapdoors can be seen as an instance of Dehn's "word search" problem. That is, from a group G, a set of generators $g_1, ..., g_k$ and a target t, determine the sequence $\{m_i\}$ given a writing $t = \prod g_{m_i}$. As such it is linked to other cryptosystems based on similar problems. The first one is a trapdoor based directly on Dehn's problem from Wagner and Magyarik [WM84], but broken later by Levy-dit-Vehel and Perret [LP10]. Another one is the family of Cayley hashes initiated by Tillich and Zémor [TZ94] and Charles, Goren, and Lauter [CLG09].

Organization of the paper. In Section 2, we recall the relevant concepts and the trapdoor constructions. In Sections 3 and 4, we investigate some properties of dwarf

Authors listed in alphabetical order: see https://www.ams.org/profession/leaders/CultureStatement04.pdf. Valerie Gilchrist and Thomas Decru are supported by grants from the National Fund for Scientific Research (F.N.R.S.) of Belgium; Christophe Petit is partly supported by EPSRC through grant number EP/V011324/1.

E-mail: thomas.decru@ulb.be (Thomas Decru), tako.fouotsa@epfl.ch (Tako Boris Fouotsa), paul.frixons@gmail.com (Paul Frixons), valerie.gilchrist@ulb.be (Valerie Gilchrist), christophe.petit@ulb.be (Christophe Petit)

Received: 2024-07-09 Accepted: 2024-09-02

matrices from the trapdoor constructions. Concrete attacks are proposed in Sections 5 and 6, and we conclude with some future work in Section 7.

2 Background

We review some of the key concepts put forward in [GSN23] for using matrix products as trapdoors. For the rest of the paper we let S_k denote the set of permutations on the integers $\{1, \dots, k\}$.

Consider some $\sigma \in S_k$. Given a set of k matrices, $\mathbf{A} = \{A_1, \dots, A_k\} \subset GL_n(\mathbb{F}_p)$, we can map σ to a matrix product as

$$\sigma\mathbf{A} := \prod_{i=1}^{k} A_{\sigma(i)}.$$

By choosing the parameters k, n, p carefully, and sampling $\mathbf{A}$ uniformly at random, we can ensure that with high probability this map is injective. That is, that the products we are mapping into are distinct for each distinct σ.

In order to build cryptography on top of this mapping, we would hope for the inverse computation to be cryptographically hard. To this end, the authors of [GSN23] first provide a very specific instance where the inversion of this map is easy, and then take advantage of these easy instances to build a trapdoor. To explore their construction, we first establish a way of determining the "size" of a matrix. To do so we define a partial order relation on the set of matrices.

Definition 1. Let $M := \{m_{i,j}\}_{i,j=1}^{n}, M' := \{m'_{i,j}\}_{i,j=1}^{n}$ be two matrices in $GL_n(\mathbb{F}_p)$. We say $M < M'$ if for all $i, j \in [1, \dots n]$, we have that $m_{i,j} < m'_{i,j}$ when expressed as integers in $[0, p-1]$.

We will use the notation $M < \alpha$, to mean that every entry in M, when expressed as an integer in $[0, p-1]$, is less than α.

Now, if the entries of the matrices of $\mathbf{A}$ are very small, then we can expect the entries of $\sigma\mathbf{A}$ to also be relatively small. In particular, we could choose our prime p so that when considering the product over the integers, it holds that $\sigma\mathbf{A} < p$. Thus we can work with these products as if they were over the integers.

In particular, barring few exceptions such as the identity matrix, we get that as we multiply more matrices together, the entries are strictly increasing. This means we expect that

$$A_{\sigma(1)} \cdots A_{\sigma(k)} > A_{\sigma(1)} \cdots A_{\sigma(k)} \cdot A_{\sigma(k)}^{-1} = A_{\sigma(1)} \cdots A_{\sigma(k-1)}.$$

This provides us with a method of recovering the permutation from the product. We give the details of this in Algorithm 1, which we call Decompose, where we assume each matrix $A_i \in \mathbf{A}$ is such that $A_i \leq \alpha$, and p is chosen such that $n^{k-1}\alpha^k < p$. This will ensure that we are working over the integers.

Notice that for Decompose (Algorithm 1) to work, *all* of the matrices in $\mathbf{A}$ have to be bounded in size. Thus, we will make a distinction between these special matrices and the rest of $GL_n(\mathbb{F}_p)$. Note, we make a point to exclude the center of $GL_n(\mathbb{F}_p)$, denoted $Z(GL_n(\mathbb{F}_p))$, as they will not be usable in the trapdoors later on. In particular, $Z(GL_n(\mathbb{F}_p)) = \{sI : s \in \mathbb{F}_p\}$.

$$\mathfrak{E} := GL_n(\mathbb{F}_p) \setminus Z(GL_n(\mathbb{F}_p)),$$
$$\mathfrak{D} := \{M \in \mathfrak{E} : M \leq \alpha\}.$$

Following the terminology introduced in [GSN23], we refer to these sets as *elves* and *dwarves* respectively. In summary, while Decompose can be run on a set of dwarf matrices,

Algorithm 1: Decompose

Input : $\mathbf{A} = \{A_1, \ldots, A_k\} \subset \mathfrak{D}$, $C = \sigma\mathbf{A}$
Output : The ordered list $\mathbf{L} = \{A_{\sigma(1)}, \cdots, A_{\sigma(k)}\}$ from which one can extract σ

```
if A = ∅ then
    return ⊥
if |A| = 1 then
    if C ∉ A then
        return ⊥
    else
        return C
for A ∈ A do
    M ← C · A^{-1}
    if M < C then
        L ← Decompose(A \ {A}, M)
        if L =⊥ then
            return ⊥
        else
            L = L||[A]
            return L
return ⊥
```

the work of [GSN23] conjectures that for random elf matrices it is not easy to recover the same information. Specifically, they claim that the following problem is hard for sufficiently large n, k, p.

Problem 1. *Let $\mathbf{A} \subset \mathfrak{E}$ be a uniformly sampled set of elf matrices. Given $\sigma\mathbf{A}$ over $\mathbb{F}_p$, compute σ.*

This leads [GSN23] to two trapdoor constructions, a *direct construction* and an *alternating construction*. The core idea to both constructions being that we will define secret maps sending a set of dwarf matrices to a set of elf matrices. By making the elf matrices public, anyone can create a product and publish it, where adversaries will not be able to recover the permutation. With knowledge of the secret mapping, however, we will be able to send the product of elves to a product of dwarves, where we can apply Decompose to recover the permutation.

2.1 Direct construction

Let $\mathbf{A} := \{A_1, \ldots A_k\} \subset \mathfrak{D}$ be a set of (secret) dwarf matrices. Choose a (secret) elf matrix, $E \in \mathfrak{E}$. We will use E to mask $\mathbf{A}$ as

$$\bar{\mathbf{A}} := \{\bar{A}_i := EA_iE^{-1}\}_{i=1}^{k}.$$

The set $\bar{\mathbf{A}}$ will serve as the public key. Now an external party can choose some permutation $\sigma \in S_k$, and compute the ciphertext

$$C := \sigma\bar{\mathbf{A}} = E(\sigma\mathbf{A})E^{-1}.$$

We can map this ciphertext back to a product of dwarf matrices since

$$E^{-1}CE = \sigma\mathbf{A},$$

from which we can apply the Decompose algorithm to recover σ.

Parameters. The authors from [GSN23] give parameters for varying security levels, which they refer to as "toy", "challenge", "recommended", and "large". We list them in the table below, where λ is their estimated security level in bits.

Table 1: Parameters from [GSN23] for the direct construction.

	λ	k	n	α	p
toy	16	9	4	2	$2^{53}+5$
challenge	64	21	8	2	$2^{167}+83$
recommended	128	35	10	2	$2^{302}+307$
large	512	99	24	2	$2^{1105}-1335$

Extra masking. In the original work of [GSN23] the authors suggest that including an extra dwarf matrix in the masking could improve security, though they do not provide concrete reasons for using it. In this variant they define their public key matrices as $\bar{A}_i := EA_iDE^{-1}$, where $D \in \mathfrak{D}$. For simplicity of exposition, we exclude this masking matrix in the proceeding sections, but will justify why including a non-trivial D does not avoid the attack later on.

2.2 Alternating construction

Let $\mathbf{A}^b := \{A_1^b, \dots A_k^b\} \subset \mathfrak{D}$, be two sets of (secret) dwarf matrices, one for each bit $b \in \{0,1\}$. Choose a set of (secret) elf matrices, $\{E_i\}_{i=0}^k \in \mathfrak{E}$. Define

$$\bar{\mathbf{A}}^b := \{\bar{A}_i^b := E_{i-1}A_i^bE_i^{-1}\}_{i=1}^k.$$

The sets $\bar{\mathbf{A}}^b$ will serve as the public keys. Now for a binary string $m \in \{0,1\}^k$, we can compute the ciphertext as

$$C = \prod_{i=1}^{k} \bar{A}_i^{m_i} = E_0\left(\prod_{i=1}^{k} A_i^{m_i}\right)E_k^{-1}.$$

We will refer to this computation with the notation $m\bar{\mathbf{A}}$ for ease of notation. Thus to map C to a product of dwarf matrices we compute $E_0^{-1}CE_k$, and then apply a variant of the Decompose algorithm to recover m. This variant is described in Algorithm 2.

Parameters. Similarly to the direct construction, the authors from [GSN23] give toy, challenge, recommended, and large parameters for the alternating construction, where λ is the estimated security level in bits. We list them in Table 2 below.

Table 2: Parameters from [GSN23] for the alternating construction.

	λ	k	n	α	p
toy	16	16	4	2	$2^{47}+5$
challenge	64	64	8	2	$2^{255}-19$
recommended	128	128	10	2	$2^{553}+549$
large	512	512	24	2	$2^{2859}+641$

Algorithm 2: Decompose

Input : $\mathbf{A}^b = \{A_1^b, \dots A_k^b\} \subset \mathfrak{D}$ for $b = 0, 1$; $C = \prod_{i=1}^{k} A_i^{m_i}$
Output : m

```
for i ∈ {1, ..., k} do
    M ← (A_i^0)^{-1} · C
    if M < C or M = Id then
        m_i ← 0
        C ← M
    else
        m_i ← 1
        C ← (A_i^1)^{-1} · C
return m = m_1||m_2|| ... ||m_k
```

3 Properties of dwarf products

We will explore some properties of dwarf matrices and dwarf matrix products. We continue to use the notation outlined in Section 2.

3.1 Dwarf determinants

We begin by considering some properties of the determinants of dwarf matrices.

We will use Hadamard matrices to obtain some upperbounds later on. Recall that a Hadamard matrix is a square matrix whose entries are all either 1 or -1, and whose columns are pairwise orthogonal. These matrices can equivalently be written using only 0 and 1, such that the columns are still orthogonal to each other. We also recall Hadamard's inequality, which states that for a matrix M with vectors v_i, we have that $|\det(M)| \leq \prod_{i=1}^{n} ||v_i||$ in the Euclidean norm.

Lemma 1. *For a dwarf matrix A it holds that $|\det A| \leq (\alpha\sqrt{n})^n$, when $\det A$ is seen as an integer in $(-p/2, p/2)$.*

Proof. This follows from Hadamard's inequality. □

In Lemma 2, we will get concrete upper bounds on the determinants of dwarf matrices, but only for the smallest values of α, namely, $\alpha = 1$ or 2. This is convenient since these choices of α coincide with the suggested parameters.

Lemma 2. *For an $n \times n$ dwarf matrix A with $\alpha \in \{1, 2\}$ and n even, the maximal value of $\det A$ is the maximal value of the determinant of the $n \times n$ Hadamard matrices with entries $\{0, 1\}$, respectively $\{-1, 1\}$.*

Proof. For $\alpha = 1$ this is immediate. For $\alpha = 2$, it suffices to see that in both cases we are computing the volume of the maximal n-dimensional hypercube with lattice points at most distance two away. □

Since not all dwarves are Hadamard matrices, the determinants of the dwarves will be much smaller than these maximal values, as can be seen from the experiments in [TV06]. Studying them, however, gives us an effective upper bound on dwarf determinants. The work of Tao and Vu [TV06, Theorem 1.1] shows that with probability tending to 1 (as n tends to infinity), the absolute value of the determinant of a random $\{1, -1\}$ Hadamard matrix is close to $\sqrt{n!}$. For $n = 8$, this means the determinant is expected to be close to 2^8.

For $\alpha = 2$ and n ranging from 1 to 10 we then get that $|\det A|$ is maximally equal to

$$2, 4, 16, 48, 160, 576, 4096, 14336, 73728, 327680.$$

There is no known closed-form expression for these values, but the first 22 can be found as `A003433` in The On-Line Encyclopedia of Integer Sequences [OEI24]. Some of the larger values of determinants cannot be attained; e.g. for $n = 3$ we cannot construct a matrix with determinant $\pm 13, \pm 14, \pm 15$. The maximum values are typically achieved by incorporating a lot of structure. The following is an example for $n = 8$ with maximal determinant:

$$\begin{pmatrix} 2 & 0 & 2 & 0 & 0 & 2 & 2 & 0 \\ 2 & 2 & 0 & 2 & 0 & 0 & 2 & 2 \\ 2 & 2 & 2 & 0 & 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 & 0 & 2 & 0 \\ 2 & 0 & 0 & 2 & 2 & 2 & 0 & 2 \\ 0 & 2 & 0 & 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 2 & 2 & 2 \end{pmatrix}.$$

3.2 Dwarf traces

The trace of a matrix is invariant under conjugation, but is not multiplicative. Thus $\mathrm{Tr}(E\sigma\mathbf{A}E^{-1}) = \mathrm{Tr}(\sigma\mathbf{A})$, and will change depending on the choice of σ. For dwarf products in general we expect that as more dwarves are multiplied to it, the trace should strictly increase. This will provide us with a test to use in our attack later on. We formalize this idea in Heuristic 1 and give both heuristic arguments and thorough experimental evidence to support it.

Heuristic 1. *described in Section 2.1.*
Let $\{A_1, \dots A_k\} \subset \mathfrak{D}$, $\sigma \in S_k$, and $m \leq k$.
Then we have that

$$Tr\left(\prod_{i=1}^{m-1} A_{\sigma(i)}\right) < Tr\left(\prod_{i=1}^{m} A_{\sigma(i)}\right)$$

with overwhelming probability.

Recall that every entry in a dwarf matrix is at most α. By considering a set of matrices, where each entry is exactly the maximum value α, we can upper bound the trace of a product of m dwarf matrices by $n^m\alpha^m$. Note that the prime is chosen to be larger than $n^{k-1}\alpha^k$. However, this upper bound is significantly larger than what occurs in practice since the bound is computed from considering a matrix with only entries equal to α, which is not itself an invertible matrix. So in practice, we expect the trace of the product to always be less than p, meaning we are working strictly over the integers.

Now, let D_α be the distribution of a variable uniformly randomly sampled in $\{0, \dots, \alpha\}$. Though dwarf matrices are required to be invertible and so there may be some bias introduced, we estimate that this bias does not have a significant impact on the results of this analysis, and we will support this claim with experimental evidence later on. Thus, we assume the elements of a dwarf matrix are sampled according to D_α. Their expected value is

$$\mu_\alpha := \frac{1}{\alpha+1}\sum_{i=0}^{\alpha} i = \alpha/2.$$

Their variance is

$$\sigma_\alpha^2 = \frac{1}{\alpha+1}\left(\sum_{i=0}^{\alpha} i^2\right) - \mu_\alpha^2 = \frac{\alpha(2\alpha+1)}{6} - (\alpha/2)^2 = \frac{\alpha(\alpha+2)}{12}.$$

We will use the following fact. Let X and Y be two random variables, and let Z be their product. Assume X and Y are independent. Then we have

$$\mu_Z = \mu_X \mu_Y \quad \text{and} \quad \sigma_Z^2 = (\sigma_X^2 + \mu_X^2)(\sigma_Y^2 + \mu_Y^2) - \mu_X^2 \mu_Y^2.$$

Entries in the product of two dwarf matrices can now be estimated as follows. Each entry is the sum of n terms, where each term is the product of two independent variables, each one distributed as D_α. Their average value is then

$$\mu_\alpha^{(2)} = n\mu_\alpha^2 = n\frac{\alpha^2}{4},$$

and their variance is

$$(\sigma_\alpha^{(2)})^2 = n\left[(\sigma_\alpha^2 + \mu_\alpha^2)(\sigma_\alpha^2 + \mu_\alpha^2) - \mu_\alpha^2\mu_\alpha^2\right] = n\alpha^2\frac{7\alpha^2 + 16\alpha + 4}{144}.$$

Entries in the product of three or more dwarf matrices are harder to estimate rigorously. We can try to iterate the previous argument, by heuristically ignoring correlations between elements of partial products. We then obtain formulae for the averages as

$$\mu_\alpha^{(k)} = n\mu_\alpha^{(k-1)}\mu_\alpha = n^{k-1}\mu_\alpha^k \tag{1}$$

and for the variances as

$$\begin{aligned}(\sigma_\alpha^{(k)})^2 &= n\left[((\sigma_\alpha^{(k-1)})^2 + (\mu_\alpha^{(k-1)})^2)(\sigma_\alpha^2 + \mu_\alpha^2) - (\mu_\alpha^{(k-1)})^2\mu_\alpha^2\right] \\ &= n\left[(\sigma_\alpha^{(k-1)})^2(\sigma_\alpha^2 + \mu_\alpha^2) + (\mu_\alpha^{(k-1)})^2\sigma_\alpha^2\right].\end{aligned} \tag{2}$$

Consider the probability that the trace of a product of $k+1$ dwarves is smaller than the trace of the product of the first k factors. In other words, letting P be the product of k dwarf matrices, and A a dwarf matrix, we are interested in when $\mathrm{Tr}(PA) - \mathrm{Tr}(P) = \mathrm{Tr}((P(A-I))$ is negative. Ignoring correlations, the expected value of the trace difference can be approximated as follows

$$\mu := \mathrm{Tr}(PA) - \mathrm{Tr}(P) = n(\mu_\alpha^{(k+1)} - \mu_\alpha^{(k)}) = n\mu_\alpha^{(k)}(n\alpha/2 - 1).$$

This average is of course positive, but we also need the variance to argue about the probability to obtain a negative value. We estimate this as follows

$$\sigma^2 = n((\sigma_\alpha^{(k+1)})^2 + (\sigma_\alpha^{(k)})^2).$$

We can then bound the probability that the trace difference is negative using Tchebychev inequality

$$\Pr\left[\mathrm{Tr}(PA) - \mathrm{Tr}(P) \leq 0\right] \leq \left(\frac{\sigma}{\mu}\right)^2. \tag{3}$$

This implies that $\epsilon(\alpha, n, k) := 1 - (\sigma/\mu)^2$ is a lower bound on the probability that $\mathrm{Tr}(PA) > \mathrm{Tr}(P)$ holds. For each set (n, k, α) of the parameters from [GSN23] for the direct construction, we computed $\epsilon(\alpha, n, k')$ where $1 \leq k' \leq k$. This data is given in Figure 1.

Experiments. In Figure 2, we computed the average trace of a product of i dwarf matrices, for $i \in [1, k]$. The averages were taken from $10,000$ samples for several parameter sets. We also computed the expected average trace from the theoretical analysis above. As can be seen, the expected average trace from the theoretical analysis matches the average trace obtained from the experiments. The traces follow a logarithmic line, hence supporting

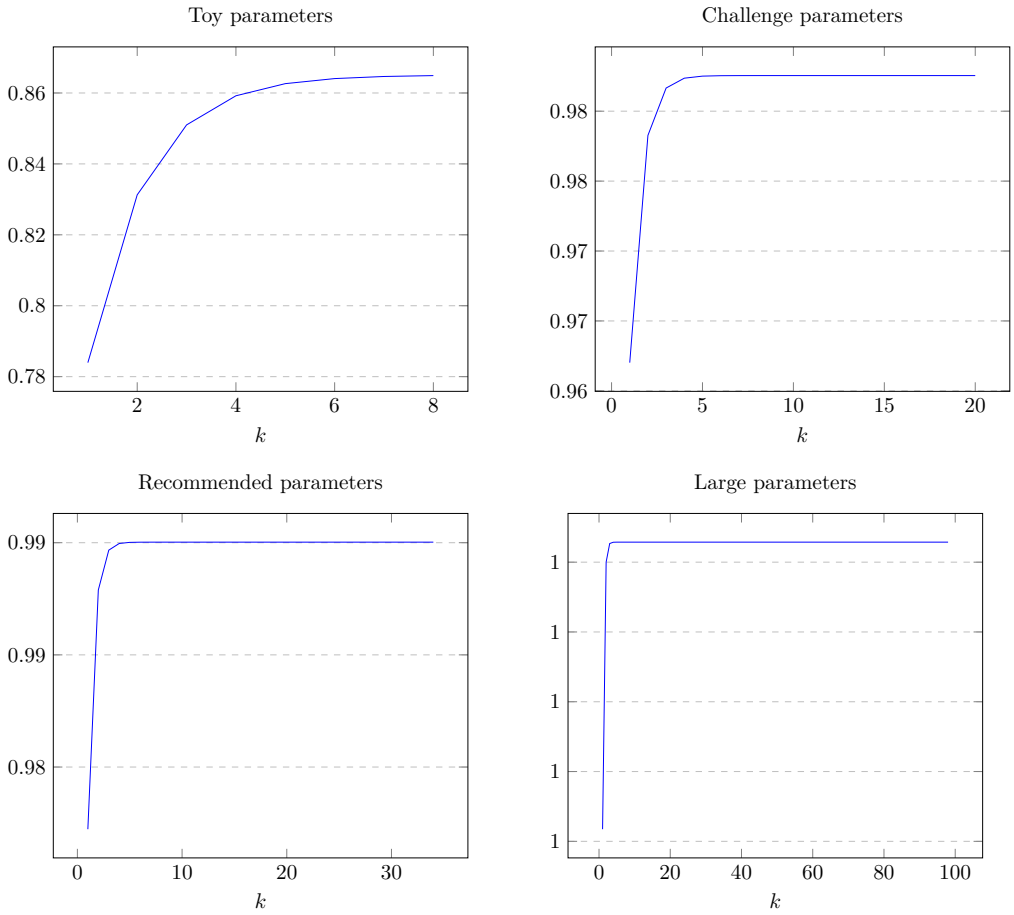

Figure 1: We plot the probabilities that the trace differences are non-negative according to Equation 3 for each set of parameters from Section 2.1.

the claim in Heuristic 1. The coloured bars (where visible) indicate the standard deviation from the experiments. We do not plot the standard deviation from the analysis since this would congest the graphs. We checked Heuristic 1 for $k = 2, 3$ using 10^6 samples for each of the toy, challenge, and recommended parameters, and found that it was true 100% of the time. Note, these experimental results do not match exactly the probabilities shown in Figure 1, which is fine since Tchebychev inequality serves only as a lower bound of the actual probabilities. Thus, these results support the claim from Heuristic 1.

The results shown in Figures 3 and 4 show how varying the values of n and α will affect the trace. In general, increasing these values will increase the average trace of the dwarf products. We also computed these values from $10,000$ samples, which were generated using the challenge parameters from the direct construction, which are shown with the circle marks. The expected trace values according to our statistical analysis are plotted using the empty square marks. We see that the lines are almost completely overlapping indicating that our analysis of the trace values was in fact accurate.

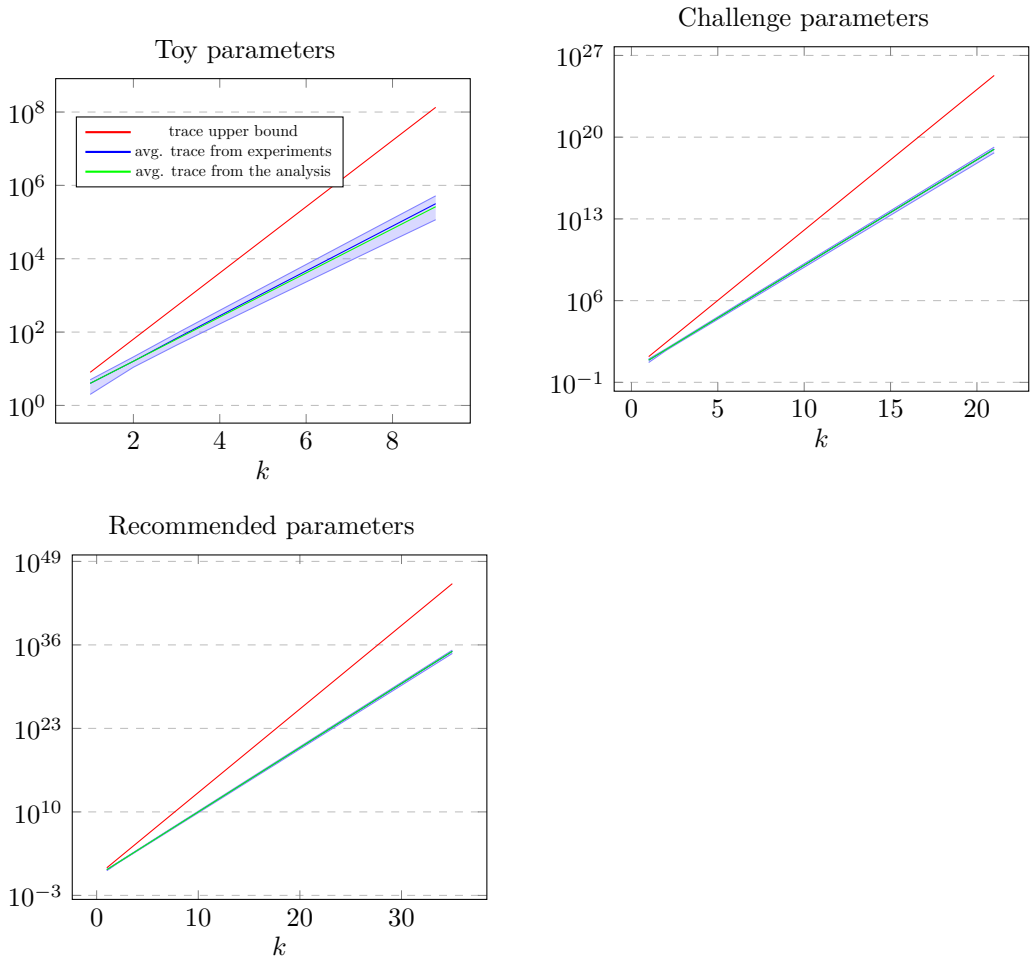

Figure 2: We plot the trace upper bound, both the experimental and theoretical (from the analysis above) average traces for products of dwarves of various parameter sets together with the maximum possible trace value.

4 Trace of inverse dwarf products

4.1 Dwarf products and inverse dwarves

Now instead of looking at how the trace changes in dwarf products, we consider how it might change when multiplying by a non-dwarf matrix in the following heuristic. Note that the inverse of a dwarf is not expected to be a dwarf.

Heuristic 2. *Let $C = \prod_{i=1}^{m} A_{\sigma(i)}$ be a product of dwarf matrices. The expected value of $Tr(C)$ is less than the expected value of $Tr(A_{\sigma(j)}^{-1} C)$ where $j \neq 1, m$.*

Note that $j = 1$ would result in a strictly smaller product, thereby giving a smaller trace. Additionally, $j = m$ results in a product of length $m - 1$ being conjugated by $A_{\sigma(m)}$. Since trace is preserved under conjugation, this would also be expected to give a smaller trace.

In our experiments, after running $10,000$ samples, we found that for the recommended parameters from the direct construction, Heuristic 2 was true 71% of the time. For the large parameter set this probability increased to 99%.

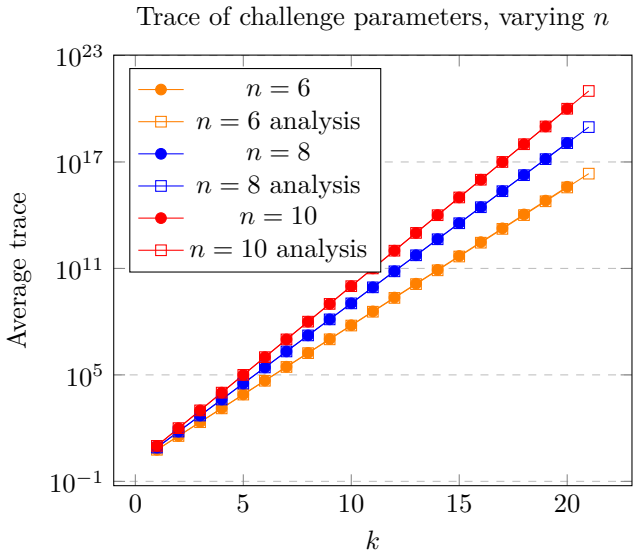

Figure 3: Average trace value for dwarf products of varying k values for the challenge parameters. The different lines indicate different n values. We plot both the experimental results and the expected results from our analysis. The lines are closely overlapping.

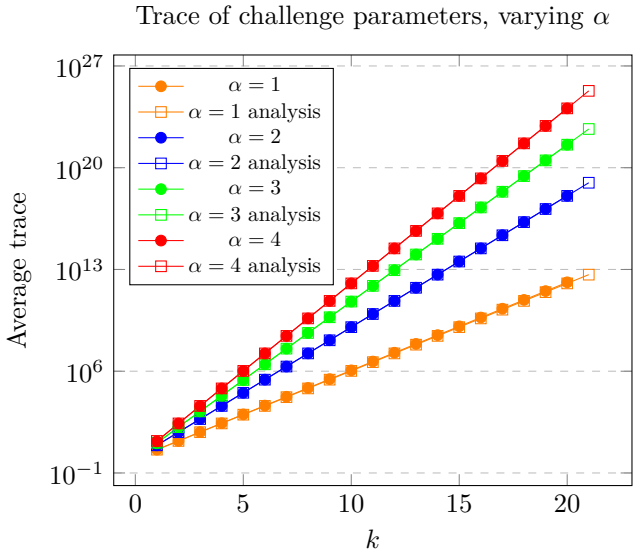

Figure 4: Average trace value for dwarf products of varying k values for the challenge parameters. The different lines indicate different α values. We plot both the experimental results and the expected results from our analysis. The lines are closely overlapping.

4.2 Estimating entries of inverse dwarves

Let D be an $n \times n$ dwarf matrix, then $D^{-1} = 1/det(D) \cdot Adj(D)$, where $Adj(D)$ denotes the adjugate of D. Recall that the entries of $Adj(D)$ are minors of D i.e. determinants of the $(n-1) \times (n-1)$ submatrices of D. Assuming the coefficients of D are distributed with mean μ and variance σ^2, we now study the expected distribution of the minors of D. A minor of size r is a sum (with signs) of $r!$ terms, where each term is a product of r coefficients in D. If these behave as independent, we then expect the minors' average values and variances to be

$$\mu_r = \begin{cases} 0 & \text{if } r \text{ is even} \\ \mu^r & \text{if } r \text{ is odd} \end{cases}, \qquad \sigma_r^2 = (r!)((\sigma^2 + \mu^2)^r - \mu^{2r}).$$

In particular, we expect the determinant to be a random variable with average and variance μ_n and σ_n^2. Coefficients of an inverse dwarf matrix times its determinant should have average μ_{n-1} and variance σ_{n-1}^2.

When considering the product of several inverse dwarves, it is the same as taking the inverse of a product of dwarves. Thus, we can apply the same analysis but where μ and σ^2 are (respectively) the mean and variance for a matrix D with larger coefficient entries. These values were already computed in Section 3.2 in Equations 1 and 2, as $\mu_\alpha^{(k)} = n^{k-1}\mu_\alpha^k$ and $(\sigma_\alpha^{(k)})^2 = n((\sigma_\alpha^{(k-1)})^2(\sigma_\alpha^2 + \mu_\alpha^2) + (\mu_\alpha^{(k-1)})^2\sigma_\alpha^2)$. This gives us a means for computing the expected entry sizes for the adjugate of a product of dwarves.

This analysis indicates that though the inverse of a dwarf matrix over $\mathbb{F}_p$ is not itself a dwarf, they still behave differently from a randomly sampled elf matrix. For example, we know that when the entries of $Adj(D)$ are considered as integers in $[0, p-1]$, they will not be small. When they are considered as integers in $[-(p-1)/2, (p-1)/2]$, however, they are in fact close to zero. In what follows, we will be looking at the *absolute value* of these matrix entries, where we consider the entry in $[-(p-1)/2, (p-1)/2]$, and multiply it by -1 if it is negative to ensure a positive value. Thus, we claim that the absolute value of the trace of $Adj(D)$ will also be small in general. We compute an upper bound for it in Lemma 3.

Lemma 3. *Let $D \in \mathfrak{D}$, so $D < \alpha$ and has dimensions $n \times n$.*
Then $|Tr(Adj(D))| \leq n\alpha^{n-1}(n-1)^{(n-1)/2}$.

Proof. The elements of $Adj(D)$ are determinants of the $(n-1) \times (n-1)$ submatrices of D. Thus, $|\mathrm{Tr}(Adj(D))| \leq |n \cdot d_{n-1}|$, where d_{n-1} bounds the determinant of dwarves of dimensions $n-1$. From Hadamard's inequality we get that

$$d_{n-1} \leq \alpha^{n-1}(n-1)^{(n-1)/2}.$$

□

We now give experimental evidence to support these claims.

Experiments. We considered 10^6 dwarf matrices, D, for each set of parameters proposed for the alternating construction. We then computed $\mathrm{Tr}(Adj(D)) = \mathrm{Tr}(\det(D) \cdot D^{-1})$ and list the average of the results in Table 3. From the analysis we should expect that the trace is 0 if n is even, and $n(\alpha/2)^{n-1} = n$ if n is odd. The experimental results (where n is always even) are close to zero, as expected.

Table 3: Toy, challenge, recommended, and large parameters from [GSN23] for the alternating construction, together with the average and standard deviation values of $\mathrm{Tr}(Adj(D))$ taken from 10^6 random samples.

	λ	n	α	p	avg. trace	stand. dev.
toy	16	4	2	$2^{47}+5$	0.03	32.19
challenge	64	8	2	$2^{255}-19$	0.05	32.21
recommended	128	10	2	$2^{553}+549$	-0.04	32.20
large	512	24	2	$2^{2859}+641$	-0.03	32.16

5 Trace attack on the direct construction

In what follows, we outline an attack on the direct construction from [GSN23], and so assume the notation and structure summarized in Section 2.1.

Given some ciphertext $C = E(\sigma\mathbf{A})E^{-1}$, and a public key $\bar{\mathbf{A}} := \{\bar{A}_1, \dots \bar{A}_k\}$, we can iteratively multiply by $(\bar{A}_i)^{-1}$, while checking the trace. If the trace is less than $\mathrm{Tr}(C)$, then we will assume that we have correctly guessed the first matrix in the product. Heuristic 2 asserts that this will likely be the case, we estimate that the probability of this being a correct guess is at least 0.99, as seen in Section 3. The issue here is that trace is invariant under conjugation, thus when we left-multiply by the inverse of the last matrix we get that

$$\mathrm{Tr}\left((\bar{A}_{\sigma(k)})^{-1} \cdot \bar{A}_{\sigma(1)} \cdots \bar{A}_{\sigma(k)}\right) = \mathrm{Tr}\left(\bar{A}_{\sigma(1)} \cdots \bar{A}_{\sigma(k-1)}\right).$$

This trace will also be smaller than $\mathrm{Tr}(C)$.

As it turns out, if we make an incorrect guess of the first matrix using this trace check, we can quickly flag it as incorrect by continuing to left-multiply by inverses. If the guess had been correct, there will always exist at least one matrix from the public key such that left-multiplying by its inverse gives a smaller trace. In the case of an incorrect guess, this quickly stops being the case, since the odds of being in one of these special cases, described above, becomes less and less likely as the product grows. Hence we can discard it. This approach is summarized in Algorithm 3.

Note that the set T_k from Algorithm 3 will contain two permutations that are equal except for the final two indices, which are permuted. This is due to the fact that once we get down to a product of two matrices, we get that $tr(\bar{A}_{k-1}\bar{A}_k) = tr(\bar{A}_k\bar{A}_{k-1})$. Determining which permutation was the correct one can be done easily by computing the corresponding products, and comparing them to the ciphertext.

We coded Algorithm 3 in Magma, which can be found at the following link:

https://github.com/vgilchri/matrix-product-attack.

We give timings for this attack on the "challenge", "recommended", and "large" parameters from [GSN23] in Table 4. The experiments were run using Magma V2.27-7 on a laptop with an Intel Dual-Core i3 at 1.1 GHz.

Table 4: Timings in seconds for attacking the direct construction, where λ was the previously estimated security in bits.

	λ	k	n	α	p	time (s)
challenge	64	21	8	2	$2^{167}+83$	0.2
recommended	128	35	10	2	$2^{302}+307$	2.8
large	512	99	24	2	$2^{1105}-1335$	915

Algorithm 3: DirectTraceAttack

Input : $C = \sigma\bar{\mathbf{A}}$, $\bar{\mathbf{A}}$
Output : σ

1 $T_0 \leftarrow \{I\}$
2 **for** $i = 1$ **to** $k - 2$ **do**
3 $T_i \leftarrow \{\}$ `/* trace-decreasing products of length` i `*/`
4 **for** $M \in T_{i-1}$ **do**
5 **for** $\bar{A}_j \in \bar{\mathbf{A}}$ **do**
6 **if** $Tr(\bar{A}_j^{-1} \cdot M^{-1} \cdot C) < Tr(M^{-1} \cdot C)$ **then**
7 $T_i \leftarrow T_i \cup \{M \cdot \bar{A}_j\}$
8 Let $\bar{A}_u, \bar{A}_v \in \bar{\mathbf{A}}$ be the two unused matrices.
9 **for** $M \in T_{k-2}$ **do**
10 **if** $M \cdot \bar{A}_u \cdot \bar{A}_v = C$ **then**
11 **return** σ such that $\bar{A}_{\sigma(1)} \cdots \bar{A}_{\sigma(k)} = M \cdot \bar{A}_u \cdot \bar{A}_v$.
12 **else if** $M \cdot \bar{A}_v \cdot \bar{A}_u = C$ **then**
13 **return** σ such that $\bar{A}_{\sigma(1)} \cdots \bar{A}_{\sigma(k)} = M \cdot \bar{A}_v \cdot \bar{A}_u$.
14 **return** $\perp$

Heuristic 3. *Given a ciphertext $C = \sigma\bar{\mathbf{A}}$, and a public key, $\bar{\mathbf{A}}$, as described in Section 2.1, Algorithm 3 can recover σ in complexity $O(k^2 n^\omega)$ under Heuristic 2, where $O(n^\omega)$ is the cost of inverting an $n \times n$-matrix.*

Looking at Algorithm 3, we see that the first loop (step 2) has length $k - 2 = O(k)$. From Heuristic 2 we assume the second (step 4) and the fourth (step 9) loop will require $O(1)$ iterations. The third loop (step 5) starts at length k, but decreases by one each time it is called (since we do not need to check matrices already in the product M). The dominating cost of step 6 is from the matrix multiplications and inversions, thus we get a total complexity of $O(k^2 n^\omega)$ for step 1 to step 8. Since the loop at step 9 requires $O(1)$ iterations, then step 9 to step 14 run in constant time $O(n^\omega)$, which does not affect the overall complexity ($O(k^2 n^\omega)$) of the whole algorithm.

Note, our experiments showed that in practice $|T_i|$ never exceeded 3 for any of the parameter sets, thereby supporting our use of Heuristic 2.

Extra masking. Recall from Section 2.1 that the authors of [GSN23] suggested that including an extra masking matrix could improve the security of the scheme. They defined their public key matrices as $\bar{A}_i := EA_iDE^{-1}$, where $D \in \mathfrak{D}$. Note, this would require a larger prime to ensure Decompose still runs, since the length of the dwarf product essentially doubles. This alternate construction, with a non-trivial choice of D, does not avoid the attack from this section. Though A_iD is not itself a dwarf matrix, the properties of the trace function being an increasing function in our context remains true. Though D can be chosen to have as large a trace as possible in the hopes of making the trace of the products larger than p, we argue this would not be an effective countermeasure since you also risk the Decompose algorithm, that is central to the trapdoor function, failing.

6 Attacking the alternating construction

Recall that in the alternating construction we have two sets of dwarf matrices, say the $\{A_i^0\}_{i=1}^k$ and the $\{A_i^1\}_{i=1}^k$. We also have elves $E_0, \ldots, E_k$. For each i, the matrix $\bar{A}^b := E_{i-1} A_i^b E_i^{-1}$ is computed. When the sender wants to encrypt a bitstring $m = m_1 ... m_k$, they compute the following product:

$$C := \prod E_{i-1} A_i^{m_i} E_i^{-1} = E_0 \big(\prod A_i^{m_i} \big) E_k^{-1}.$$

In Section 6.1 we outline an information leak that impacts the entropy of the scheme, then in Section 6.2 we describe a meet-in-the-middle attack that we will use as a baseline for comparison against another new attack in Section 6.3.

6.1 Recovering the determinant of dwarf matrices

In what follows we will assume that n is much smaller than k, which is true for every suggested parameter set of the protocol. Recall, we have matrices $\bar{A}_i^b$ which – when combined – we can turn into conjugate matrices as follows:

$$\bar{A}_i^0 (\bar{A}_i^1)^{-1} = E_{i-1} A_i^0 (A_i^1)^{-1} E_{i-1}^{-1}.$$

Following [GSN23, Remark 7], A_i^0 and A_i^1 are chosen to have the same determinant as to not leak information. We know that A_i^0 is a dwarf such that it has small entries. This is unfortunately not true for $(A_i^1)^{-1}$, since it is the inverse of a matrix with small entries, so when seen as a matrix over $\mathbb{Z}$, its entries can be huge. However, as described in Section 4, we can construct the inverse of a matrix through its determinant and its adjugate; i.e. $(A_i^1)^{-1} = \frac{1}{\det A_i^1} \cdot \mathrm{adj} A_i^1$. We also know that $\det A_i^1$ is small since we can upperbound it in terms of n and α like in Lemma 1. Let us write $d_i := \det A_i^0 = \det A_i^1$ as well as $\bar{d}_i := \det \bar{A}_i^0 = \det \bar{A}_i^1$.

Now consider

$$d_i \cdot \bar{A}_i^0 (\bar{A}_i^1)^{-1} = E_{i-1} A_i^0 \cdot \mathrm{adj}(A_i^1) E_{i-1}^{-1}.$$

Even though the adjugate will not have coefficients bounded by α, it *will* have coefficients much smaller than p; i.e. the entries are bounded from above by entry $n-1$ of the `A003433` sequence, as seen in Section 3.1. Recall, we also assume that n is much smaller than k, which is true for every suggested parameter set of the protocol. Furthermore, conjugate matrices keep their characteristic polynomial (which includes the trace as well as the determinant).

With the public information, we can compute the characteristic polynomial of the matrix $\bar{A}_i^0 (\bar{A}_i^1)^{-1}$ (as an element of $\mathbb{F}_p[x]$) and use lattice techniques to find the coefficients of this polynomial (as elements of $\mathbb{Z}$). Indeed, write $\sum_{i=0}^n c_i x^i$ for the characteristic polynomial (seen with coefficients in $[0, \ldots, p-1]$), and consider the lattice generated by the rows of the following matrix:

$$\Lambda = \begin{pmatrix} c_0 & c_1 & c_2 & \ldots & c_{n-1} \\ p & 0 & 0 & \ldots & 0 \\ 0 & p & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & p \end{pmatrix}.$$

Remark that $c_0 = 1$ since $\bar{A}_i^0$ and $\bar{A}_i^1$ have the same determinant. When multiplying $\bar{A}_i^0 (\bar{A}_i^1)^{-1}$ with $\bar{d}_i$, we get the matrix $\bar{A}_i^0 Adj(\bar{A}_i^1)$ with characteristic polynomial $\bar{d}_i \sum_{i=0}^n c_i x^i$, whose entries are expected to be small relative to p, given that the coefficients of both $\bar{A}_i^0$ and $Adj(\bar{A}_i^1)$ are small relative to p. Now, the vector $\vec{v} = (\bar{d}_i c_0, \ldots, \bar{d}_i c_{n-1})$ is

in Λ, so we can expect it to be a relatively short vector. The first entry of this short vector will be the determinant of A_i^1 up to sign, since $c_0 = 1$. From this, one can also easily deduce the coefficients of the characteristic polynomials of $A_i^0 \cdot (A_i^1)^{-1}$ and $A_i^1 \cdot (A_i^0)^{-1}$, seen as elements in $\mathbb{Q}$.

How likely is the vector $\vec{v}$ to be the shortest vector in Λ? Due to the construction of the characteristic polynomial, its first entry can achieve the largest possible value so if we bound this from above by the aforementioned maximal value we get

$$\begin{aligned} ||\vec{v}|| &= ||(\bar{d}_i c_0, \quad \dots \quad , \bar{d}_i c_{n-1})|| \\ &\leq ||(\bar{d}_i c_0, \quad \dots \quad , \bar{d}_i c_0)|| \\ &= ||(\bar{d}_i, \quad \dots \quad , \bar{d}_i)|| \\ &= \sqrt{n} \cdot \bar{d}_i \\ &\leq \sqrt{n} \cdot (\alpha\sqrt{n})^n \\ &= \alpha^n \sqrt{n}^{n+1}. \end{aligned}$$

On the other hand, we can consider the Minkowski upper bound for the shortest vector λ_1 in Λ:

$$\begin{aligned} ||\lambda_1|| &\leq \sqrt{n} \cdot \left(\sqrt{\det(\Lambda^T \Lambda)}\right)^{1/n} \\ &= \sqrt{n} \cdot \left(p^{n-1} \sqrt{(p^2 + (\sum_{i=0}^{n-1} c_i^2)}\right)^{1/n} \\ &\leq \sqrt{n} \cdot \left(p^{n-1} \sqrt{p^2 + np^2}\right)^{1/n} \\ &= \sqrt{n} \cdot p \cdot (\sqrt{n+1})^{1/n}. \end{aligned}$$

Now if we assume $p \approx \alpha^k n^{k-1}$ as in the protocol we can simplify this to

$$||\lambda_1|| \leq \alpha^k n^k.$$

Even though it is clear that $\vec{v}$ is short compared to this bound, and we know that $n << k$, the lattice Λ is not constructed at random so we cannot conclude that $\vec{v}$ is likely to be the shortest vector. In fact, for small values of p and n one can easily construct counterexamples to this statement. For realistic parameters however, only one exceptional case appears heuristically. This is the case where all entries of $\vec{v}$ share a common factor over $\mathbb{Z}$, in which case a shortest vector algorithm will only get $\vec{v}$ divided by this common factor. It is hard to determine what the odds of this happening are exactly, given that the c_i are not drawn uniformly at random and will depend on the (small) parameter α. Generically, we expect every c_i to be divisible by a prime ℓ_j with probability ℓ_j^{-1}, so all of them will be divisible by ℓ_j simultaneously with a probability of ℓ_j^{-n}. This probability is heavily dominated by $\ell_j = 2$ of course, and in practice we see that this is the case as well for all realistic parameters. Even for the challenge parameters, it means we can determine the determinant of the used dwarf matrices in the public key with over 99% accuracy. When swapping to the recommended parameters, this turns into 99.9% already. This can be seen from the code provided at

https://github.com/vgilchri/matrix-product-attack.

Since the determinant of the dwarf matrices leaks, this means the key generation can be unlucky and have a pair of dwarf matrices A_i^0, A_i^1 with very large determinant, although the chances of this happening are rather slim.[1]

[1]Remark that this is also noticeable in the timing of the key generation: sampling random elements

Countermeasures. One possible countermeasure would be to only sample dwarves from a set $\mathfrak{D} \cap \mathfrak{F}$, where $\mathfrak{F}$ is a set of matrices with fixed determinant (up to sign). The choice of $\mathfrak{F}$ would of course require enough entropy for $\mathfrak{D} \cap \mathfrak{F}$. One suggestion would be $\mathfrak{F} = SL_n(\mathbb{F}_p)$, although depending on α and n, better options may be available. Apart from this, any type of security reduction would still need to take into account information leaking from the characteristic polynomials of $A_i^0 \cdot (A_i^1)^{-1}$ and $A_i^1 \cdot (A_i^0)^{-1}$, making it a lot harder to argue why they would be indistinguishable from random for example.

6.2 Meet-in-the-middle

The parameter selection from [GSN23, Section 5.5] indicates that the security in bits, λ, is taken to be equal to k. This security claim comes from analyzing the complexity of a brute force attack, in that a message of length k, will have 2^k possibilities to check.

We consider running a meet-in-the-middle attack. Recall that we are given the list of public matrices, $\{\bar{A}_i^0\}_{i=1}^k$, $\{\bar{A}_i^1\}_{i=1}^k$, the product $C := \prod \bar{A}_i^{m_i}$, and are asked to recover the bitstring $m = m_1 ... m_k$. We create a list of guesses for the first $k/2$ bits, and a list of guesses for the last $k/2$ bits. For each of these guesses, we compute the corresponding matrix product and store those in two separate lists, which we denote $\{D_i\}$ and $\{F_i\}$ respectively. What we would like is to find a $D' \in \{D_i\}$ and $F' \in \{F_i\}$ such that $D' \cdot F' = C$, and thus such that $D' = C(F')^{-1}$. So in reality we will be looking for collisions between the lists $\{D_i\}$ and $\{CF_i^{-1}\}$.

This attack has time complexity $O(2^{k/2})$, which halves the bit complexity of the overall scheme. The memory complexity is also $O(2^{k/2})$, but algorithms such as van Oorschot-Wiener [vOW99] can be used to improve upon these memory requirements. We use this as a baseline with which to compare the attack in the proceeding section.

6.3 Trace attack

The naïve approach to applying the trace attack from Section 5 would be to compute a new product from a bit string, d, of the form

$$D := \prod E_{i-1} A_i^{d_i} E_i^{-1} = E_0 \big(\prod A_i^{d_i} \big) E_k^{-1}.$$

Then we can compute CD^{-1} to obtain a product of the form

$$CD^{-1} = E_0 \prod A_i^{m_i} \prod (A_i^{d_i})^{-1} E_0^{-1}.$$

This is a product of dwarves and inverse dwarves, conjugated by one elf. Recall that the determinant can be assumed to be known because of Section 6.1, so we can rewrite this product with adjugate matrices instead. The hope, thus, would be that since the product is conjugated by an elf, we can use the trace to gain some knowledge about the secret, m.

We will be able to use the same overall idea to gain information about the secret using the trace, but will need to consider the absolute value of the trace instead. This time we have more variance in the size of the entries, unlike in Section 5 where we were guaranteed to have entries with a fixed upper bound. We will need to guess several bits at once in order to see noticeable differences in the absolute value of the trace. Another issue is that since the product of matrices now has $2k$ matrices, it is likely that the combined trace of the product will be larger than p. So in order for the attack to run we will first need to guess some bits of m, and then we can run the attack to recover the remaining bits of m.

We will begin with some initial guess d, and take note of the absolute value of the trace $|\mathrm{Tr}(CD^{-1})|$. Then, we will adjust d by switching the last *step* bits of d with every

from $\mathfrak{D}$ until a second one with the same exceptional determinant is found can take an egregious amount of time. Even for more common determinants, sampling the keys this way is a nontrivial task.

Algorithm 4: AlternatingTraceAttack

Input : $\bar{\mathbf{A}}^0$,$\bar{\mathbf{A}}^1$, $step$, $C = m\bar{\mathbf{A}}$
Output : m

$str \leftarrow$ AllBitStrings(length=$step$)
$T \leftarrow \{str\}$ /* track "good" candidates */
$prev \leftarrow p - 1 - \epsilon$ /* track previous trace for comparison */
for $i = 1$ **to** $k'/step$ **do**
  for $b \in T$ **do**
    $traces \leftarrow \{\}$
    for $s \in str$ **do**
      $D_s \leftarrow (b||s||0 \cdots 0)\bar{\mathbf{A}}$ /* add zeros until length k */
      $t_s \leftarrow Abs(Tr(CD_s^{-1}))$
      if $t_s < prev$ **then**
        $traces \leftarrow traces \cup \{t_s\}$
  $T \leftarrow \{s : t_s < min(traces) + \epsilon\}$
  $prev \leftarrow min(traces) + \epsilon$
return $m \in T$

possible bit string of length $step$, and compare all of the traces. We expect there to be a big difference in trace size between correct and incorrect guesses, as seen in Heuristic 2. Thus, we keep any bit strings that achieve the minimum trace value, or close to the minimum trace value (we denote this interval using ϵ). We iterate until we have some candidates for m.

We outline the attack in Algorithm 4, where we would like to recover k' bits of m, assuming we have already guessed the first $k - k'$ bits. Recall that the notation $m\bar{\mathbf{A}}$ refers to computing a product of matrices using $\bar{\mathbf{A}}^0$ and $\bar{\mathbf{A}}^1$ that relies on m, as outlined in Section 2.2. We use ϵ to denote some "wiggle room", that is, we want to keep any candidates that give a trace that is *close* to the minimum, even if not exactly the minimum. We will use AllBitStrings(length=$step$) to denote a function that lists all bitstrings of length $step$. Note also we will use $prev$ to track what the previous product's trace was in order to keep only products of decreasing trace. It is initialized to be as large as possible since the first product does not have any trace to be compared to.

We coded the attack in Magma, which can be found at the following link:

https://github.com/vgilchri/matrix-product-attack.

We see that the first loop has length $k'/step$. The second loop has length upper bounded by 2^{step}, but in practice is far smaller. The third loop has length exactly 2^{step}. The dominating subroutine within the loops is the matrix multiplications, which has complexity $O(n^\omega)$, where ω depends on the choice of algorithm used. Based on experimental evidence, we estimate that $step$ can be chosen between $k'/8$ and $k'/4$, with the latter being more expensive but having a higher likelihood of success. In total this means that the complexity is $O(2^{2 \cdot step} n^\omega)$, which experimentally can range between $O(2^{k'/4} n^\omega)$ and $O(2^{k'/2} n^\omega)$. This gives a time-accuracy trade-off for the attack.

Recall, however, that this attack can only be used as a subroutine to a larger attack. The first step of the larger attack constitutes guessing the first $k - k'$ bits of the message. For each such guess, we run Algorithm 4. This leaves us with a total bit complexity of $\lambda' = k - k' + 2 \cdot step$. Using the analysis from Section 4.2, we can estimate how large to choose k' such that the entries in the largest possible product are less than p.

Heuristic 4. *Algorithm 4 is expected to return the correct bitstring whenever the length of the input product, k, satisfies $n^k(n^{k-1})^{n-1} < p$.*

Applying the attack from Section 6.1 we can assume to know the determinant, and so we need only estimate the size of the adjugate, and not the inverse. An entry in a product of k dwarves (when $\alpha = 2$) is expected to be around n^{k-1}, and taking the adjugate of this gives entries that are determinants of $(n-1) \times (n-1)$ matrices. This gives a total expected size of $(n^{k-1})^{n-1}$ (see Section 4.2 for more on estimating the size of minors). Thus when we multiply the k product of dwarves with the k product of adjugate dwarves, we get that an entry will be around $n \cdot n^{k-1} \cdot (n^{k-1})^{n-1} = n^k(n^{k-1})^{n-1}$. Setting this less than p provides us with an estimate for k'. We list the estimates of k' for each parameter set from the alternating construction in Table 5.

Table 5: Estimated choices of k' for each parameter set from the alternating construction.

	n	k	k'
toy	4	16	6
challenge	8	64	11
recommended	10	128	17
large	24	512	26

We see that these estimates of k' lead to an attack that performs worse than the meet-in-the-middle attack from Section 6.2, but the attack still shows that [GSN23] had a lower security than what was claimed. Furthermore, Algorithm 4 requires very little memory, which is the main caveat of the meet-in-the-middle attack. In order to further improve upon this attack, we outline an experimental approach to choosing k'.

Experimentally increasing k'. In what follows we chose k' experimentally such that a product of $2k'$ matrices did not exceed p more than 99% of the time, leaving us with bigger k' values that were competitive with the meet-in-the-middle attack from Section 6.2.

For our experimental analysis and in order to better estimate the resource requirements and accuracy of our attack on larger parameter sets, we created additional parameter sets following the description in [GSN23] for 24, 32, 40, 48, and 56 bits of security. Following [GSN23], we take $\alpha = 2$ for all the sets. Since we need that $k \geq \lambda$, we simply take them equal. We choose n depending on λ and then take the smallest prime such that $p > \alpha^k n^{k-1}$. The new parameter sets of the form (λ, k, n, p) we obtained are

$$(24, 24, 4, \alpha^k n^{k-1} + 25),$$
$$(32, 32, 5, \alpha^k n^{k-1} + 47),$$
$$(40, 40, 6, \alpha^k n^{k-1} + 35),$$
$$(48, 48, 6, \alpha^k n^{k-1} + 91),$$
$$(56, 56, 6, \alpha^k n^{k-1} + 331).$$

We list the choices of k', and some experimental results in Table 6. The results listed in the table are averages from running the attack 100 times using Magma V2.28-5 on an Intel Xeon CPU E5-2630v2 at 2.60GHz.

The new estimated bit complexities, λ', are often the same as (or sometimes worse than) the complexity of the meet-in-the-middle attack from Section 6.2, however with some fine-tuning of the parameters selected in Algorithm 4, some improvements may be possible. At the moment we have chosen *step* to try to optimize the success probability, but choosing smaller *step* values would lead to improved complexities over meet-in-the-middle and still succeed for a significant portion of instances.

Table 6: Experimental results for partial attack on the alternating construction. λ was the previously estimated security in bits and λ' is the newly estimated bit security, k' is the number of bits that are recovered in the attack from m, "step" refers to the number of bits guessed at once, timings are given in seconds, and the probability of success is given as a percent.

λ	λ'	k'	step	time (s)	success (%)
24	12	16	2	0.15	96
	16		4	0.30	100
32	14	24	3	0.20	97
	20		6	1.71	100
40	20	28	4	1.25	100
48	24	32	4	9.27	100
56	32	36	6	114.87	100
64	40	32	4	11.4	100
128	80	64	8	3418.96	55
	96		16	10172.06	100

Countermeasures. One possible countermeasure would be to reduce the size of the prime to as small as possible such that Decompose still runs. This would save back some bits of security, but does not avoid the attack entirely.

The other obvious countermeasure is to increase the size of k. As mentioned, we are estimating the new security value as $\lambda' = k - (3/4)k'$, so in order to reach λ bits of security, we would need $k = \lambda + (3/4)k'$. For example, this means that for 64 bits of security, $k = 88$.

Remark. In [GSN23, p.17] the authors outline how the alternating construction could be used in a KEM. In particular, the scheme requires taking the final matrix modulo a small prime since Decompose is not necessary. This construction of KEM would not be affected by any of the attacks outlined in this paper.

7 Conclusion

We have shown that the constructions outlined in [GSN23] are not secure by giving a full message recovery attack on the direct construction, running in polynomial time, and detailing several other weaknesses in the alternating construction. We expect these attacks cannot easily be avoided with simple countermeasures.

We believe the attack from Section 6.3 may be an interesting example where machine learning could be used to make a more adaptive attack that makes use of previous (failed) guesses, and optimizes some of the bounds used such as ϵ and k'. It may also be combined with the ideas from Section 6.1 to decrease the size of the product and thus allow for a larger k' value. Future work could also consider how the attacks detailed in this paper may apply to other non-commutative objects, such as tensors, or other word problem based cryptosystems.

References

[CLG09] Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of CRYPTOLOGY*, 22(1):93–113, 2009.

[GSN23] Remi Geraud-Stewart and David Naccache. New public-key cryptosystem blueprints using matrix products in $\mathbb{F}_p$. Cryptology ePrint Archive, Paper 2023/1745, 2023. URL: https://eprint.iacr.org/2023/1745.

[LP10] Françoise Levy-dit-Vehel and Ludovic Perret. Security analysis of word problem-based cryptosystems. *Des. Codes Cryptogr.*, 54(1):29–41, 2010. URL: https://doi.org/10.1007/s10623-009-9307-x, doi:10.1007/S10623-009-9307-X.

[OEI24] OEIS Foundation Inc., 2024. Entry A003433 in The On-Line Encyclopedia of Integer Sequences, https://oeis.org/A003433.

[TV06] Terence Tao and Van Vu. On random ± 1 matrices: singularity and determinant. *Random Structures Algorithms*, 28(1):1–23, 2006. doi:10.1002/rsa.20109.

[TZ94] Jean-Pierre Tillich and Gilles Zémor. Hashing with sl_2. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994. doi:10.1007/3-540-48658-5_5.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptol.*, 12(1):1–28, 1999. doi:10.1007/PL00003816.

[WM84] Neal R. Wagner and Marianne R. Magyarik. A public key cryptosystem based on the word problem. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1984. doi:10.1007/3-540-39568-7_3.