Generating cryptographically-strong random lattice bases

…and…

recognizing rotations of $\mathbb{Z}^n$

https://eprint.iacr.org/2021/154.pdf

Tamar Lichter Blanks and Stephen D. Miller

Department of Mathematics

Rutgers University

PQCrypto 2021

The Twelfth International Conference on Post-Quantum Cryptography

Structure of talk

- Background
- Generating random matrices in $GL(n, \mathbb{Z})$
- Experiments testing generation methods
- Applications to the DRS NIST submission

Structure of talk

- Background
- Generating random matrices in $GL(n, \mathbb{Z})$
- Experiments testing generation methods
- Applications to the DRS NIST submission

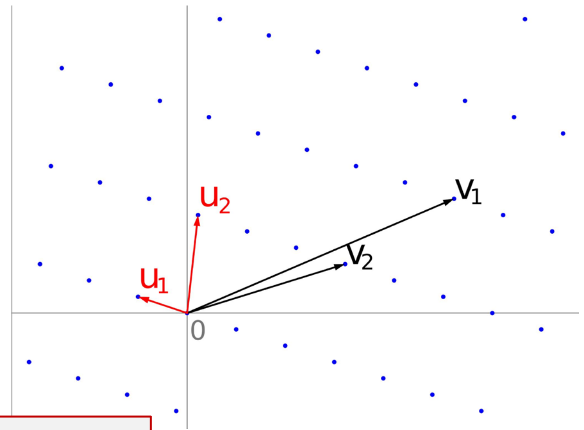
Overview

- Hard lattice problems are widely popular for post-quantum crypto
- Need average-case hard distributions for these problems.
  - Can we efficiently recognize rotations of the $\mathbb{Z}^n$ lattice?
- We compare methods to sample random bases (keys)
  - $\cong$ elements of $GL(n, \mathbb{Z})$
- We find existing weaknesses in:
  - multiplying elementary matrices ("unipotents")
  - DRS NIST submission method
- Suggest other, stronger methods

Lattices, bases, and integer matrices

- Lattice $\Lambda \subset \mathbb{R}^n$: the *integral* linear combinations of some basis $\mathcal{B} = \{b_1, \ldots, b_n\}$
  - $\Lambda = \{m_1 b_1 + m_2 b_2 + \cdots + m_n b_n \mid m_1, m_2, \ldots, m_n \in \mathbb{Z}\}$
  - Discrete set of points which can be added, subtracted.
- In example here, can take $b_1 = u_1$ and $b_2 = u_2$.
- However, the basis *not* unique: $\{v_1, v_2\}$ is also a basis
- Here $v_1 = -5u_1 + 2u_2$ and $v_2 = -2u_1 + 1u_2$
- The coefficients form an integral matrix $\begin{pmatrix} -5 & 2 \\ -2 & 1 \end{pmatrix}$ whose inverse must also be integral (since can write u_1, u_2 in terms of v_1, v_2).
- **Upshot:** different basis correspond to matrices in

$$GL(n, \mathbb{Z}) = \{\text{all integral matrices whose inverse is invertible}\}$$
$$= \{\text{integral matrices with determinant} \pm 1\}$$



The basis $\{u_1, u_2\}$ is easier to work with than $\{v_1, v_2\}$.

Crypto: needs way to convert from easy to hard

but **what if** no hard bases exist?

Recognizing rotations of $\mathbb{Z}^n$

- Not even known whether rotations of the standard integral lattice $\mathbb{Z}^n$ have hard bases!
- Lenstra-Silverberg problem:
  - Let B be the $n \times n$ matrix whose rows are the basis vectors $\{b_1, \dots, b_n\}$
    - For example, spans $\mathbb{Z}^n$ if and only if $B \in GL(n, \mathbb{Z})$.
  - Define *Gram matrix* $G = BB^t$

Problem 2a (Decision version).

Given a positive-definite integral matrix G, efficiently determine whether or not there is some $M \in GL(n, \mathbb{Z})$ such that $G = MM^t$.

Problem 2b (Search version).

If so, efficiently find such a matrix $M \in GL(n, \mathbb{Z})$.

Problem 3 (Average case version of Problem 2b).

Given a random matrix $M \in GL(n, \mathbb{Z})$ drawn with respect to the probability density p, efficiently recover M from MM^t (up to signed permutations of the columns) with high probability.

- Note: $B = MR$ and R orthogonal if and only if $BB^t = MM^t$.
  - Thus problem asks whether lattice spanned by $\{b_1, \dots, b_n\}$ is a rotation of the integer lattice.

Structure of talk

- Background
- Generating random matrices in $GL(n, \mathbb{Z})$
- Experiments testing generation methods
- Applications to the DRS NIST submission

Generating random matrices: brute force

Algorithm 1.

For each $1 \leq i, j \leq n$ sample $m_{i,j} \in \mathbb{Z} \cap [-T, T]$ at random.
Let $M = (m_{ij})$.
Discard and repeat if $\det(M) \neq \pm 1$, otherwise
return M.

- Pros: definitely samples $GL(n, \mathbb{Z})$ uniformly
- Cons: very *unlikely* that $\det(M) = \pm 1$ … so *very* inefficient.

- Only useful for small n (used as a subroutine in later algorithms)

Generating random matrices: RandomSLnZ

Algorithm 2 (Random products of unipotents, such as Magma's `RandomSLNZ`).

Input: a size bound b and word length ℓ.
Return: a random product $\gamma_1 \cdots \gamma_\ell$, where each γ_k is chosen i.i.d. uniformly among all $n \times n$ matrices of the form $I_n + xE_{i,j}$, with $i \neq j$ and $x \in \mathbb{Z} \cap [-b, b]$.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & x & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Here x = -1, 0, or 1
- Pros: Easy to implement, provably exhausts $SL(n, \mathbb{Z})$
- Cons: As we shall see, produces biased output
  - consequently, easy to break
  - Requires large word length ℓ

Generating random matrices: smaller groups

Algorithm 3 (Random products of smaller matrices).

Input: a word length ℓ and fixed dimension $2 \leq d < n$ for which one can uniformly[a] sample $GL(d, \mathbb{Z})$ matrices in a fixed box.
Return: a random product $\gamma_1 \cdots \gamma_\ell$ in which each $\gamma_j \in GL(n, \mathbb{Z})$ is a matrix of the form $\Phi_{k_1,\ldots,k_d}(\gamma^{(d)})$, where $\gamma^{(d)}$ is a uniformly sampled random element of $GL(d, \mathbb{Z})$ in the fixed box mentioned above, and $\{k_1, \ldots, k_d\}$ is a uniformly sampled random subset of $\{1, \ldots, n\}$ containing d elements.

[a] More generally, one can consider non-uniform distributions as well.

$$\begin{pmatrix} -1 & 0 & -2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -5 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ -3 & 0 & 5 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 & 3 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -5 & 0 & -4 & -1 & 0 \\ 3 & 0 & 4 & -2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 3 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -2 & 0 & 0 & 4 & -5 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} -2 & 0 & 5 & 22 & -11 \\ 0 & 1 & 0 & 0 & 0 \\ 11 & 0 & -15 & -80 & 47 \\ 4 & 0 & 4 & 1 & 7 \\ 17 & 0 & -29 & -143 & 79 \end{pmatrix}$$

- Pros: samples more uniformly than previous as d grows.
- Cons: similar to previous Algorithm 2 when $d = 2$
  - Brute force step makes large d prohibitive

- Interpolates between Algorithms 1 and 2

Generating random matrices: random bottom

Algorithm 4 (slight modification of a suggestion of Joseph Silverman).

Uniformly sample random integers $m_{i,j} \in [-T, T]$, for $2 \leq i \leq n$ and $1 \leq j \leq n$, until the n determinants in (2.4) share no common factor.
Use the euclidean algorithm to find integers $m_{11}, \ldots, m_{1n}$ such that $\det((m_{ij})) = \pm 1$, the sign chosen uniformly at random.
Use least-squares to find the linear combination $\sum_{i \geq 2}^{n} c_i [m_{i1} \cdots m_{in}]$ closest to $[m_{11} \cdots m_{1n}]$, and let $\widetilde{c_i}$ denote an integer nearest to c_i.
Return: the matrix M whose top row is

$$[m_{11} \cdots m_{1n}] - \sum_{i \geq 2}^{n} \widetilde{c_i} [m_{i1} \cdots m_{in}]$$

and whose i-th row (for $i \geq 2$) is $[m_{i1} \cdots m_{in}]$.

1. Fills out the bottom $n - 1$ rows at random
2. It's very likely there exists a top row making the whole matrix in $GL(n, \mathbb{Z})$
3. Find such a row, reduce it to make it smaller

Generating random matrices: well-known HNF

Algorithm 5 (via Hermite Normal Form).

Create a uniformly distributed $m \times n$ matrix B, with $m \geq n$ and entries uniformly chosen in $\mathbb{Z} \cap [-T, T]$.
Decompose B in a Hermite normal form $B = UM$, where $M \in GL(n, \mathbb{Z})$ and $U = (u_{ij})$ has no nonzero entries with $i < j$.
Return: M.

- This is folklore, commonly used
- **Surprising fact:** in practice, very often produces the *same output* as Algorithm 4!
- Pros: good randomness properties
- Cons: Algorithm 4 seems to slightly outperform it

Structure of talk

- Background
- Generating random matrices in $GL(n, \mathbb{Z})$
- Experiments testing generation methods
- Applications to the DRS NIST submission

Experiments: How we evaluated the algorithms

- Given a matrix B generated by an algorithm
- attempt to solve Problem 2b by these steps:
  1. LLL on Gram matrix $G = BB^t$ (Magma's L2 implementation)
  2. BKZ with block sizes 3, 4, and 5 (seems not to matter much)
  3. Success declared if all output basis vectors have norm 1

     (since we have a rotation of $\mathbb{Z}^n$ lattice)

Experiments: Magma's RandomSLnZ

Algorithm 2 (Random products of unipotents, such as Magma's `RandomSLNZ`**).**

Input: a size bound b and word length ℓ.
Return: a random product $\gamma_1 \cdots \gamma_\ell$, where each γ_k is chosen i.i.d. uniformly among all $n \times n$ matrices of the form $I_n + xE_{i,j}$, with $i \neq j$ and $x \in \mathbb{Z} \cap [-b, b]$.

$n = \dim(\Lambda)$	entry size	ℓ
886	$[2^{25}, 2^{32}]$	55,000
1486	$[2^{14}, 2^{20}]$	55,000

- This shows weakness in high dimensions
- Special structure of matrices are likely to blame.

Experiments: Algorithm 3

Algorithm 3 (Random products of smaller matrices).

Input: a word length ℓ and fixed dimension $2 \leq d < n$ for which one can uniformly[a] sample $GL(d, \mathbb{Z})$ matrices in a fixed box.
Return: a random product $\gamma_1 \cdots \gamma_\ell$ in which each $\gamma_j \in GL(n, \mathbb{Z})$ is a matrix of the form $\Phi_{k_1,\ldots,k_d}(\gamma^{(d)})$, where $\gamma^{(d)}$ is a uniformly sampled random element of $GL(d, \mathbb{Z})$ in the fixed box mentioned above, and $\{k_1, \ldots, k_d\}$ is a uniformly sampled random subset of $\{1, \ldots, n\}$ containing d elements.

[a] More generally, one can consider non-uniform distributions as well.

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
200	2	1	4000	6.03607	12.7988	×
200	2	2	1500	1.29248	18.5329	✓
200	2	2	2000	7.86583	22.2151	×
200	2	3	1000	0.5	27.0875	×
200	2	3	2000	23.521	41.5678	×
200	2	10	500	2.04373	38.7179	✓
200	2	10	700	7.943	49.0346	×
200	3	1	1000	2.04373	11.3283	✓
200	3	1	1500	7.66619	17.1312	×
200	3	1	2000	13.0661	20.8768	×
200	3	2	500	3.27729	18.4087	✓
200	3	2	600	4.89232	24.111	×
200	3	2	1000	13.0585	34.0625	×
200	4	1	500	3.66096	12.2277	✓
200	4	2	300	0.5	24.2424	✓
200	4	2	400	1.79248	26.6452	×

key: n=lattice dimension, d=size of smaller embedded matrices, T=bound on embedded matrix entries, ℓ=length of the product of smaller matrices.

- Generally one observes failure for large enough ℓ
- Tricky to compare between different d
- Seems to not fully depend on entry size.

Experiments: Algorithm 3

Algorithm 3 (Random products of smaller matrices).

Input: a word length ℓ and fixed dimension $2 \leq d < n$ for which one can uniformly[a] sample $GL(d,\mathbb{Z})$ matrices in a fixed box.
Return: a random product $\gamma_1 \cdots \gamma_\ell$ in which each $\gamma_j \in GL(n,\mathbb{Z})$ is a matrix of the form $\Phi_{k_1,\ldots,k_d}(\gamma^{(d)})$, where $\gamma^{(d)}$ is a uniformly sampled random element of $GL(d,\mathbb{Z})$ in the fixed box mentioned above, and $\{k_1,\ldots,k_d\}$ is a uniformly sampled random subset of $\{1,\ldots,n\}$ containing d elements.

[a] More generally, one can consider non-uniform distributions as well.

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
500	2	1	4000	0.	5.90085	✓
500	2	1	8000	3.41009	10.7467	✓
500	2	1	10000	7.08508	12.7447	✓
500	2	1	15000	12.6617	18.5326	✓
500	2	1	20000	18.0246	24.5732	×
500	2	2	4000	4.21731	18.587	✓
500	2	2	6000	12.3467	28.7882	×
500	2	2	8000	18.87	35.7267	×
500	2	2	10000	28.5508	45.8028	×
500	2	3	2000	0.	19.0752	✓
500	2	3	3000	7.38752	32.9895	✓
500	2	3	4000	16.9325	40.9656	×
500	2	10	1000	0.	30.3755	✓
500	2	10	2000	11.9964	61.5006	×

key: n=lattice dimension, d=size of smaller embedded matrices, T=bound on embedded matrix entries, ℓ=length of the product of smaller matrices.

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
500	3	1	1000	0.	5.39761	✓
500	3	1	2000	1.29248	9.164	✓
500	3	1	3000	2.37744	13.9903	✓
500	3	1	4000	8.43829	17.4593	✓
500	3	1	5000	14.1789	21.528	✓
500	3	1	6000	18.3878	25.2578	×
500	3	1	7000	20.5646	29.287	×
500	3	2	1000	0.	15.551	✓
500	3	2	2000	3.24593	33.0945	✓
500	3	2	3000	23.5966	43.7986	×
500	3	3	1000	0.	28.1575	✓
500	3	3	2000	16.6455	53.1806	×
500	3	3	3000	41.3371	83.9486	×
500	4	1	1000	0.	9.85319	✓
500	4	1	2000	8.11356	18.9434	✓
500	4	1	3000	19.1019	26.9836	✓
500	4	1	4000	24.4869	35.6328	×
500	4	1	5000	26.6804	44.3982	×
500	4	1	6000	40.5944	53.3654	×
500	4	2	1000	6.29272	33.4373	✓
500	4	2	2000	33.6181	63.3469	×

Experiments: Algorithm 3

Algorithm 3 (Random products of smaller matrices).

Input: a word length ℓ and fixed dimension $2 \leq d < n$ for which one can uniformly[a] sample $GL(d, \mathbb{Z})$ matrices in a fixed box.
Return: a random product $\gamma_1 \cdots \gamma_\ell$ in which each $\gamma_j \in GL(n, \mathbb{Z})$ is a matrix of the form $\Phi_{k_1,\ldots,k_d}(\gamma^{(d)})$, where $\gamma^{(d)}$ is a uniformly sampled random element of $GL(d, \mathbb{Z})$ in the fixed box mentioned above, and $\{k_1, \ldots, k_d\}$ is a uniformly sampled random subset of $\{1, \ldots, n\}$ containing d elements.

[a] More generally, one can consider non-uniform distributions as well.

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
886	2	1	3000	0	3.49434	✓
886	2	1	4000	0	3.80735	✓
886	2	1	5000	0	4.40207	✓
886	2	1	6000	0	5.30459	✓
886	2	1	7000	0	6.16923	✓
886	2	1	8000	0	6.90754	✓
886	2	1	9000	1	7.58371	✓
886	2	1	10000	2.37744	8.05954	✓
886	2	1	15000	5.46942	11.2176	✓
886	2	1	20000	8.6594	14.5837	✓
886	2	1	25000	10.884	18.035	✓
886	2	1	30000	15.0082	21.0333	✓
886	2	1	35000	17.6964	24.8408	✓
886	2	1	40000	20.7706	28.3888	✓
886	2	1	45000	24.484	30.6745	✓
886	2	1	50000	25.7401	34.0742	×

key: n=lattice dimension, d=size of smaller embedded matrices, T=bound on embedded matrix entries, ℓ=length of the product of smaller matrices.

- Here $d = 2$, so similar to RandomSLnZ
- Works very well in high dimensions
- Like RandomSLnZ, fails for long-enough products

Experiments: Algorithm 4 (Silverman/HNF)

n	T	shortest row length (in bits)	longest row length (in bits)	found M?
100	1	2.91645	4.65757	✓
100	3	4.14501	5.81034	✓
100	4	4.50141	6.20496	✓
100	10	5.64183	7.15018	✓
100	50	7.99332	9.77546	✓
100	1	2.91645	4.65757	✓
110	1	2.98864	4.54902	×
120	1	3.03304	4.77441	×
125	1	3.09491	4.93979	×
150	1	3.12396	5.09738	×
200	1	3.42899	5.32597	×
200	2	4.23584	6.42421	×
200	3	4.72766	6.82899	×
200	4	5.06529	7.41803	×

key: n=lattice dimension, T=bound on matrix entries in bottom $n-1$ rows.

- Much stronger than others (never found M when $n \geq 110$).
- We recommend using Algorithm 4
- Use of smaller matrices (with products) in Algorithms 2 and 3 seems to be a weakness.

Structure of talk

- Background
- Generating random matrices in $GL(n,\mathbb{Z})$
- Experiments testing generation methods
- Applications to the DRS NIST submission

Where to look for weaknesses?

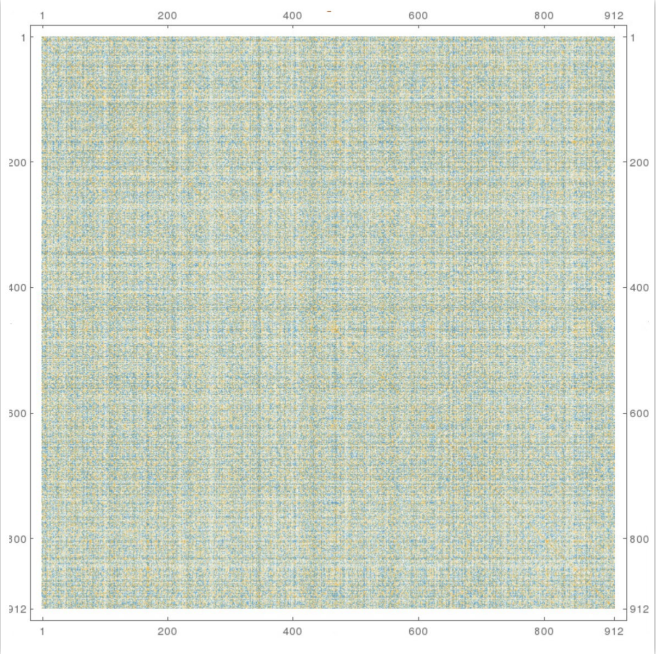
- Experiments indicate product structure is a weakness
- The DRS NIST PQC proposal includes a random lattice basis construction
  - Creates a matrix in $GL(n, \mathbb{Z})$ as a product (next slide)
  - We studied this matrix construction (not the cryptosystem itself)

DRS details

- Rough parameters:

Bit security	n
128	912
192	1160
256	1518

Product has *banded* structure (non-random)

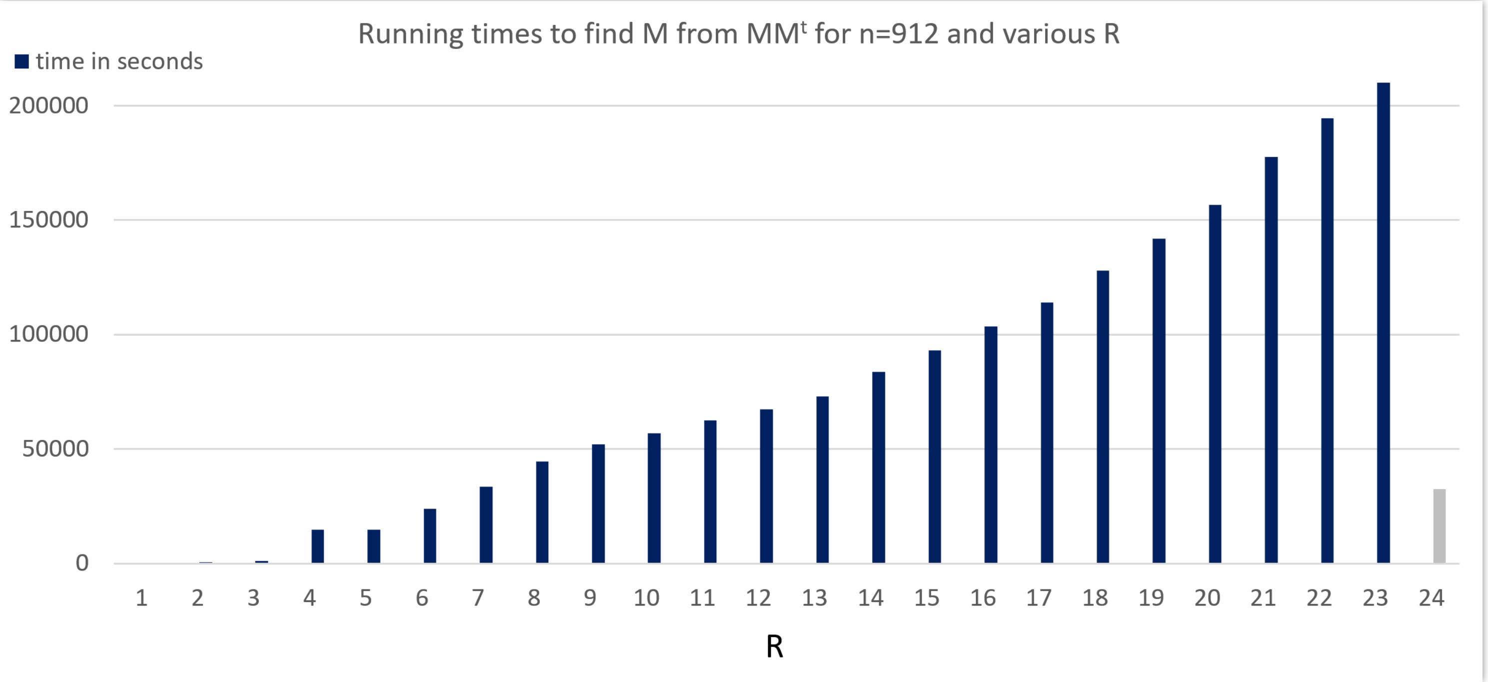


- The matrix in $GL(n, \mathbb{Z})$ is constructed as a product of 49 factors:
- Define a 2x2 matrix $A_{\pm} = \begin{pmatrix} 1 & \pm 1 \\ \pm 1 & 2 \end{pmatrix}$
- Sample γ_i uniformly among random block diagonal matrices $diag(A_{\pm}, A_{\pm}, \ldots, A_{\pm})$
  (with $n/2$ independent choices of signs)
- Sample P_i uniformly among random permutation matrices
- DRS matrix has form $M = P_1 \gamma_1 P_2 \gamma_2 P_3 \cdots P_R \gamma_R P_{R+1}$ $(R = 24)$
  where each P_i and γ_i are constructed independently as above.

Experiments: DRS NIST-PQC submission

Outcomes

- Completely recovered M in dimensions 1160 and 1518 (192- and 256-bit settings)
- Nearly recovered M in dimension 912
  - Can with 47 or fewer factors ($R \leq 23$)
- This is only for the matrix generation
  - and only on the "recognizing $\mathbb{Z}^n$" problem



Conclusions

- Not all basis generation methods are equal
- For the "recognizing $\mathbb{Z}^n$" problem, we find serious weaknesses with random unipotents
  - Magma's RandomSLnZ
- Also find problems when small matrices (e.g., 2x2) are used repeatedly
  - Basis generation method in DRS
- Methods which fill out whole matrix at once appear much stronger
  - Silverman's bottom-up method
  - Hermite Normal Form
  - (These 2 are fairly equivalent)

https://eprint.iacr.org/2021/154.pdf