The cool and the cruel:
separating hard parts of LWE secrets

Niklas Nolte[*1], Mohamed Malhou[*1], Emily Wenger[*1], Samuel Stevens[2],
Cathy Li[3], François Charton[†1], and Kristin Lauter[†1]

[1] FAIR, Meta
[2] Ohio State University
[3] University of Chicago

Abstract. Sparse binary LWE secrets are under consideration for standardization for Homomorphic Encryption and its applications to private computation [18]. Known attacks on sparse binary LWE secrets include the sparse dual attack [4] and the hybrid sparse dual-meet in the middle attack [17], which requires significant memory. In this paper, we provide a new statistical attack with low memory requirement. The attack relies on some initial lattice reduction. The key observation is that, after lattice reduction is applied to the rows of a q-ary-like embedded random matrix **A**, the entries with high variance are concentrated in the early columns of the extracted matrix. This allows us to separate out the "hard part" of the LWE secret. We can first solve the sub-problem of finding the "cruel" bits of the secret in the early columns, and then find the remaining "cool" bits in linear time. We use statistical techniques to distinguish distributions to identify both the cruel and the cool bits of the secret. We provide concrete attack timings for recovering secrets in dimensions $n = 256$, 512, and 768. For the lattice reduction stage, we leverage recent improvements in lattice reduction (flatter [32]) applied in parallel. We also apply our new attack in the RLWE setting for 2-power cyclotomic rings, showing that these RLWE instances are much more vulnerable to this attack than LWE.

Keywords: Learning With Errors, R-LWE, Sparse secrets

1   Introduction

Lattice-based cryptosystems are attractive candidates for Post-Quantum Cryptography (PQC), standardized by NIST in the 5-year PQC competition (2017-2022) and by the Homomorphic Encryption community [2,15]. Special parameter choices, such as binary, ternary, binomial, or Gaussian secrets, small error, or small *sparse* secrets, are often made to improve efficiency or functionality. Homomorphic Encryption (HE) implementations routinely assume small error and sparse binary

---

[*] Equal Contribution
[†] Co-Senior authors

or ternary secrets [2,14], and the NIST-standardized Kyber schemes assume small secret and small error (binomial distribution) [15].

Although these implementation choices may weaken security of the lattice-based cryptosystems, surprisingly few concrete results and benchmarks exist to quantify the actual time and resources required to attack systems with sparse secrets and small error. Over the last few decades, the lattice cryptography community has developed theoretical predictions of time and resources needed to attack lattice-based cryptosystems in general by studying and improving lattice reduction algorithms, such as LLL, BKZ, BKZ2.0 with various strategies such as sieving, early abort etc. ([23,16,34]). To estimate the performance of these algorithms, the community relies on the LWE Estimator [8], but these estimates are often inaccurate for special parameter choices and for parameter sizes that can be tested in practice. While some attacks on sparse secrets have been proposed, such as the Sparse dual attack [4] and the Hybrid Sparse Dual Meet-in-the-Middle attack [17], general benchmarks for measuring attacks on small sparse secrets have not yet been developed. Existing benchmarks, such as the Darmstadt Lattice Challenges [11] are not relevant, since they do not cover such parameter choices.

Our Contribution. Given the potentially widespread adoption of lattice cryptosystems with sparse, small secrets, more work is needed to assess their hardness. To that end, we provide a new tool for attacking sparse small secrets, based on the observation that lattice reduction with certain parameters separates secrets into "cruel bits", corresponding to unreduced data which is hard to analyze, and "cool bits" that can be recovered in linear time once the hard bits are known. This leads us to define an attack on sparse small secrets, in three stages:

1. initial partial lattice reduction in parallel;
2. brute force recovery of a small number of hard, cruel bits;
3. statistical recovery of cool bits, and the secret.

We provide concrete experimental results for our attack, demonstrating its efficacy on LWE instances with sparse binary secrets in dimension $n \leq 768$. Furthermore, we show how this attack applies to both LWE and RLWE instances, with RLWE instances being much more vulnerable. We believe this attack should be taken into account whenever sparse secrets are being proposed, since a significant fraction of secrets are vulnerable to it (see Table 6), and we hope it provides a helpful practical benchmark for future research.

High-Level Attack Idea. Our key observation is that, for an LWE instance $(\mathbf{A} \in \mathbb{Z}_q^{m \times n}, b \in \mathbb{Z}_q^m)$, lattice reduction of q-ary embedded matrices of the form

$$\mathbf{\Lambda} = \begin{pmatrix} 0 & q\mathbf{I}_n \\ \omega\mathbf{I}_m & \mathbf{A} \end{pmatrix}, \tag{1}$$

produces a reduced matrix $\mathbf{A}'$ with a non-uniform distribution of variances over its columns, see Section 3 for details. Specifically, the variance of the first n_u columns of $\mathbf{A}'$ is unaffected by lattice reduction. We call the bits of the secret

in the first n_u columns "cruel" (unreduced). The other $n_r = n - n_u$ columns of the matrix have significantly reduced variance, and we call the secret bits corresponding to these columns the "cool" bits. The number of unreduced and reduced columns of $\mathbf{A}'$, n_u and n_r, depends on the overall lattice reduction quality and the penalty parameter ω, which controls a trade-off between reduction and the error variance after reduction.

Our attack leverages this observation to reduce the original LWE problem to one with higher error on only the "cruel" entries in $\mathbf{A}$, by essentially ignoring the "cool" bits. If the error introduced by the ignored bits is sufficiently small, a secret guess s^* where only the first n_u bits are correct can be identified with sufficient statistics. The remaining (cool) bits can then be recovered one by one.

Table 1. *Successful LWE secret recovery attacks leveraging our cruel/cool observation for various n/q settings. The timings do not include time for parallelized lattice reduction.*

n	$\log_2 q$	Hamming Weight	Time (Sec)
256	12	12	3,865
512	28	12	2,417
	41	60	376
768	35	12	1,291

Table 1 presents parameter settings and timings for successful recovery of sparse binary secrets, and demonstrates the benefits of cruel-cool bit separation in non-trivial LWE settings (see Table 6 for additional results). In these examples, Hamming weights were selected so that secret recovery from reduced matrices only takes a small amount of compute. We choose those settings because the lattice reduction step is theoretically least well understood, whereas the rest of the attack can be understood better theoretically (see Section 5) and does not need extensive experimentation. For example, for dimension $n = 512$ and $\log_2 q = 41$, our attack recovers binary secrets with Hamming weight 60 in about 6 minutes on one GPU. This does not include the time spent on lattice reduction to prepare the data, which is about 12 hours/matrix (see Table 4). Table 1 also shows successful secret recovery for larger dimensions, such as $n = 768$ and $\log_2 q = 35$, for binary secrets with Hamming weight 12, in roughly 22 minutes using 20 GPUs.

RLWE attack. Our attack can also be applied in the Ring-LWE setting. When the ring is a 2-power cyclotomic ring defined by the polynomial $x^n + 1$, where $n = 2^k$, then the RLWE samples corresponding to a polynomial RLWE sample, $(a(x), b(x) = a(x) \cdot s(x) + e(x))$ can be described via a skew-circulant matrix, A_{circ}; see Section 7 for details. When the matrix has this structure, we can rotate the "cruel bits" around (without redoing any lattice reduction) by inspecting samples from only certain indices of circulant matrices. This greatly increases our chances of recovering the secret and makes attacking the 2-power cyclotomic RLWE problem clearly easier than generic LWE. We give concrete timings for the RLWE setting, estimate the average speed-up over LWE, and show the improved success rate in Section 7.

Outline of the paper. Section 2 describes related work. Section 3 presents our attack in detail. Section 4 explains the lattice reduction step for processing LWE samples to use in the attack. Section 5 contains the statistical analysis of the attack, including an estimate for number of samples needed. Section 6 presents our concrete secret recovery results and displays performance of other attacks. Section 7 adapts the attack to the RLWE setting and shows the improvement there. See [redacted link] for the Python code for running the attack.

2 Related Work

Albrecht et al [5] classify attacks on LWE broadly into primal [26,27,6] and dual [28,4] attacks. Some primal attacks reduce search-LWE (see Section 3 for definition) to the unique shortest vector problem (uSVP) and solve this via lattice reduction [7]. Others employ a combination of lattice reduction and Bounded Distance Decoding (BDD) [26]. The goal of dual attacks is to solve Decision-LWE (see Section 3) by reducing it to the Shortest Integer Solution (SIS) problem [1], further reduced into the problem of finding short vectors in the dual lattice defined by $\{x \in \mathbb{Z}_q^m | Ax = 0 \mod q\}$.

Hybrid attacks. The notion of hybrid attacks was introduced in 2007 by [22], who proposed combining lattice reduction and meet-in-the-middle (MITM) attacks against NTRUEncrypt. [12] extended this idea to binary-error LWE. A collection of such hybrid attacks have been proposed for different LWE settings. [10] provide a succinct overview in their Table 1. [33] succeeded in applying a hybrid attack to settings of sparse ternary secrets with small error via a primal attack strategy. [4] was the first work to combine the notions of hybrid attacks in the dual lattice for sparse binary and ternary secrets and small error. The attack benefits from optimizing a trade-off between success probability by guessing parts of the secret and performing the costly lattice reduction on the now lower-dimensional lattice. [17] improved upon [4] by using MITM for the guessing part. [21] optimized further by recognizing an inefficiency in the matrix multiplication during the guessing and considered small, non-sparse secrets. [10] extended this line of attack to arbitrary secrets and find that "hybrid dual attacks [usually] outperform [non-hybrid] dual attacks regardless of the secret distribution".

Our work. In contrast to (but not necessarily incompatible with) work in hybrid attacks, our attack performs the lattice reduction on a full-sized lattice using different subsets of available LWE samples. Our lattice is endowed with a scaling (or penalty) parameter ω. The scaling is similar to the scaling used in [9,4], and its utility is discussed in Section 4. Our key insight, the non-uniformity over coordinates in the reduced basis, naturally splits the problem into two parts (in this case the "cruel" and "cool" bits), reminiscent of a hybrid attack.

3 The Attack

The Learning With Errors (LWE) problem, first introduced by Regev [31], can be stated in 2 forms. The Search-LWE problem is: given a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$,

Table 2. *Notation used in this paper.*

Symbol	Description
q	The modulus of the LWE problem considered
n	Problem dimension (the dimension of vectors $\mathbf{a}$ and $\mathbf{s}$)
n_u	The number of unreduced (aka cruel) entries in $\mathbf{a}$.
n_r	Number of reduced (aka cool) entries in $\mathbf{a}$. $n = n_u + n_r$
$\mathbf{s}$	The unknown secret, used to construct $b = \mathbf{a} \cdot \mathbf{s} + e$
$\mathbf{s}^*$	A candidate secret, not necessarily correct
h	The Hamming weight of the secret (number of 1 bits)
h_u	The Hamming weight of the n_u unreduced bits of the secret
h_r	The Hamming weight of the n_r reduced bits of the secret ($h = h_u + h_r$)
σ_u	The standard deviation of unreduced entries in $\mathbf{a}$ (equal to $\frac{q}{\sqrt{12}}$)
σ_r	The standard deviation of reduced entries in $\mathbf{a}$
σ_e	The standard deviation of error (amplified by reduction)
$\sigma(x)$	The standard deviation of the random variable x
ρ	The reduction factor of pre-processing, i.e. the ratio $\frac{\sigma(\mathbf{RA})}{\sigma(\mathbf{A})}$
m	The number of LWE samples (reduced or unreduced)
ω	The penalty used during reduction
$a : b$	integer range $[a, b]$, used for indices
$\mathbf{X}_{*,i}$	ith column of $\mathbf{X}$

a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, and an error vector $\mathbf{e} \in \mathbb{Z}_q^m$ (usually sampled from some small error distribution), find $\mathbf{s}$, given $\mathbf{A}$ and

$$\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \mod q, \tag{2}$$

where m denotes the number of LWE samples, n is the lattice dimension, and q is the modulus. In the related Decision-LWE problem, the objective is not to find the secret vector $\mathbf{s}$, but to distinguish between two distributions: given a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$, decide whether $\mathbf{b}$ is drawn from the distribution $\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \mod q$ for some secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ and error vector $\mathbf{e} \in \mathbb{Z}_q^m$, or whether $\mathbf{b}$ is drawn from the uniform random distribution over $\mathbb{Z}_q^m$. [31] showed that for q of size polynomial in n, there is a reduction from Search-LWE to Decision-LWE. See Table 2 for notation used in the paper.

We introduce a new attack on the LWE problem for sparse binary secrets, i.e. $\mathbf{s} \in \{0, 1\}^n$, with h, the Hamming weight (number of 1s in $\mathbf{s}$), small. The attack leverages an observation about the shape of reduced LWE matrices to separate the hard and easy parts of the secret. It works as follows.

Attack Part 1: Lattice Reduction. We begin by applying lattice reduction to (the rows of) an embedding $\mathbf{\Lambda} \in \mathbb{Z}_q^{(m+n) \times (m+n)}$ of the data:

$$\mathbf{\Lambda} = \begin{pmatrix} 0 & q\mathbf{I}_n \\ \omega\mathbf{I}_m & \mathbf{A} \end{pmatrix} \tag{3}$$

The reduction finds a linear transformation $[\mathbf{C}, \mathbf{R}]$ such that $\mathbf{\Lambda}^r = [\mathbf{C}, \mathbf{R}]\,\mathbf{\Lambda} = [\omega\mathbf{R}\mathbf{I}_m, q\mathbf{C}\mathbf{I}_n + \mathbf{R}\mathbf{A}]$. We extract the transformation matrix $\mathbf{R} = \mathbf{\Lambda}^r_{*,0:m}/\omega$, which corresponds to the row and column operations performed to transform the matrix A into its reduced form. $\mathbf{R}$ is applied to both $\mathbf{A}$ and $\mathbf{b}$ to create reduced sample

pairs. For ease of notation, we will refer to **RA** as **A** or "reduced **A**" and **Rb** as **b** throughout the text, and otherwise specify "original" pairs.

Reduction algorithms trade off the error on the reduced LWE pairs for the norm of the rows in reduced **A**. The "penalty parameter" ω controls this trade-off, where higher ω causes less norm and variance reduction, but also less error amplification. Details of lattice reduction can be found in Section 4.
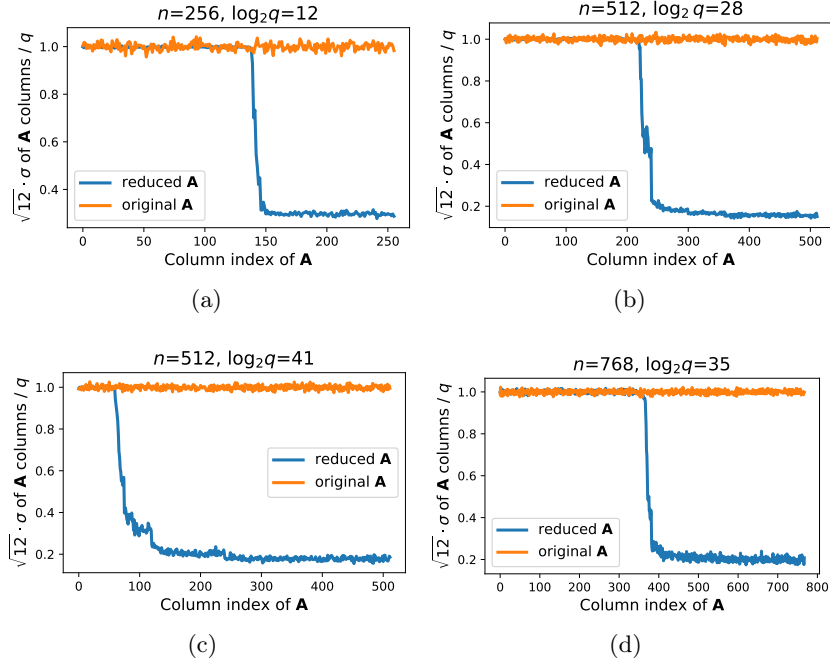

Fig. 1. *The standard deviation of elements within each column of the* **A** *matrix before and after reduction (and extraction from the q-ary embedding) for various n/q settings. The first n_u unreduced columnms (left half of the figures) correspond to the "cruel" bits of the secret, while the remaining $n_r = n - n_u$ are the "cool" bits. This phenomenon is distinct from the "z-shape" exhibited by the Gram-Schmidt orthogonalized rows of a q-ary lattice before and after lattice reduction [22,3] (see Appendix A.3 for details).*

Attack Part 2: Identify Cruel and Cool Bits. The key observation of this attack is that the resulting reduced **A** samples all share a reduction pattern:

> **Key observation:** *The first n_u elements of each sample vector $\mathbf{A}_{*,0:n_u}$ remain unreduced and retain their uniform distribution over $\mathbb{Z}_q$, and all subsequent $n_r = n - n_u$ elements of $\mathbf{A}_{*,n_u:n}$ are heavily reduced.*

To illustrate this key observation, we plot the standard deviation over columns in **A** against the column index in Figure 1 for various n and $\log_2 q$ settings. This observation of cruel and cool bits inspires the following hypothesis:

> **Hypothesis:** For any row $\mathbf{a}$ of $\mathbf{A}$, the n_r last entries contribute comparatively little to the overall dot product $\mathbf{a} \cdot \mathbf{s}$. Thus, for a sparse enough secret, the Decision-LWE may be solvable without correctly guessing the n_r secret bits.

In other words, this shape resulting from lattice reduction means we only need to solve LWE on a smaller dimension $n_u < n$. In essence, *we separate out the hard part of the LWE secret at the cost of increased error*. We validate this hypothesis in the remainder of this paper.

Attack Part 3: Secret Guessing. Next, we leverage statistical properties of the cool and cruel regions to guess the secret. Figure 2 shows a histogram of the distribution of the residuals $\mathbf{a} \cdot \mathbf{s} - b \mod q$ (in blue) and $\mathbf{a} \cdot \mathbf{s}^* - b \mod q$ (in orange), where $\mathbf{s}^*$ shares the first n_u with $\mathbf{s}$ and has all other n_r elements set to 0. Green corresponds to a random guess for the secret with equal hamming weight. The orange and the green distributions are distinct, enabling us to detect the correctness of the first n_u bits in the secret without knowing the others. Given the size of the error on the reduced LWE problem, so far we have not found a more efficient attack for guessing the first n_u secret bits than brute force. One could think of using one of the hybrid attack strategies outlined in the related work on the cruel part, but we leave this for future work.
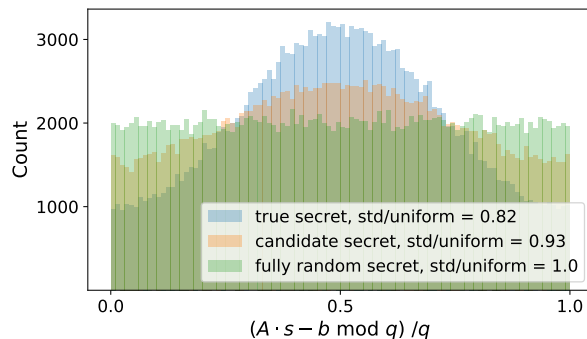

Fig. 2. *Histogram of 4 million samples of the residual $(\mathbf{a} \cdot \mathbf{s}^* - b) \mod q$ (centered and normalized) for different secret guesses $\mathbf{s}^*$ ($n = 512$, $\log_2 q = 41$, $h = 20$). The three histograms correspond to three different guesses. Blue corresponds to the true secret, orange to the secret candidate that has only the first $n_u = 75$ bits correct and green to a random secret with $h = 20$. The numbers in the legend correspond to the sample standard deviation relative to the uniform standard deviation $\frac{q}{\sqrt{12}}$.*

Given a secret guess $\mathbf{s}^*$ with correct n_u cruel bits, the n_r remaining cool bits can be recovered easily, given enough samples. One fast method is greedy recovery, presented in Algorithm 1. It works as follows: start with a candidate secret $\mathbf{s}^*$ which shares the first n_u cruel bits with $\mathbf{s}$ and has all other n_r bits set to 0. Starting with the first cool bit, flip that bit of $\mathbf{s}^*$ to 1. Then calculate if this

7

flip increases the standard deviation of the resulting $\mathbf{a} \cdot \mathbf{s}^* - b \mod q$ (over m reduced samples $(\mathbf{a}, b)$). If so, flip it back to 0 and move to the next bit. Note the standard deviation here is a practical proxy for a proper statistical test. Section 3.1 expands on this. Intuitively, greedy recovery works because every correct bit flipped to 1 will reduce the orange distribution in Figure 2 closer to the blue and every wrong one will introduce more noise, sending it closer to the green.

Algorithm 1 Greedy recovery of cool bits

1: **for** $i \in n - n_r : n$ **do**
2:      $\mathbf{s}^*[i] \leftarrow 0$
3:      $\sigma_0 \leftarrow \sigma(\mathbf{a} \cdot \mathbf{s}^* - b \mod q)$
4:      $\mathbf{s}^*[i] \leftarrow 1$
5:      $\sigma_1 \leftarrow \sigma(\mathbf{a} \cdot \mathbf{s}^* - b \mod q)$
6:      **if** $\sigma_0 \leq \sigma_1$ **then**
7:          $\mathbf{s}^*[i] \leftarrow 0$
8:      **else**
9:          $\mathbf{s}^*[i] \leftarrow 1$
10:      **end if**
11: **end for**

Otherwise, one can employ more principled gradient-free optimization methods, like simulated annealing. Annealing works better with limited data, but is slower. The best choice depends on the setting. All methods work efficiently due to the reduced variance of the corresponding entries in $\mathbf{A}$.

3.1 Practical considerations

Implementation. As described in detail in Section 4, we sample views of the $4 \cdot n$ sized lattice defined by the LWE samples and reduce them individually, forming a large set of reduced LWE samples for use in the attack, following [25,24]. After identifying the cool and cruel bits, secret recovery runs in two parts: first, guessing the cruel secret bits on the the reduced LWE problem with large error, then the recovery of the full secret via Algorithm 1.

We use brute force enumeration for cruel secret bit recovery. All code is implemented in Python, and we employ GPU acceleration via `pytorch` [29]. Batches of secret candidates are continuously sent to the device and evaluated on a suitable amount of data to distinguish the residuals $\mathbf{A} \cdot \mathbf{s}^* - \mathbf{b} \mod q$ if the true (part of the) secret was found, see the orange histogram in Figure 2. Since our brute force is perfectly parallelizable, we can distribute the work to an arbitrary number of GPUs or CPUs. Candidate secrets are enumerated in ascending Hamming weight, and a record of the top-k candidates, according to a metric defined below, are evaluated at constant intervals throughout the enumeration. Evaluation involves running Algorithm 1 and then checking if the resulting secret produces nearly correct $(\mathbf{A}, \mathbf{b})$ pairs. The evaluation frequency can be traded off against the amount of data used during enumeration and k, the number of candidates kept in the top-k set.

Candidate Evaluation. The metric by which candidate secrets are sorted can involve any test that distinguishes from uniformity in the residual, but in practice it must be fast. Most non-parametric tests like Kolmogorov-Smirnov or Kuiper involve calculating a Cumulative Distribution Function (CDF) and thus sort the data, which is slow. A likelihood ratio test is generally more powerful, but evaluating such a test on a distribution modulo q involves evaluation over multiple "wraps" around the modulus. Under certain assumptions, detailed in Section 5, the distribution we aim to distinguish from uniform is a mod-Gaussian involving an infinite sum:

$$f(x) \propto \sum_{k=-\infty}^{\infty} \mathcal{N}(x; \mu + k \cdot q, \sigma^2). \tag{4}$$

Even with suitable truncation, the runtime of a likelihood ratio test will at best be $O(N \cdot K)$ with N samples and truncation to K elements of the sum. [21] also explores a way of specifically distinguishing modular Gaussians via a periodic aggregation metric $Y = \sum_{i=1}^{N} \exp(2\pi i X_i)$, see their Algorithm 2. We thought, since both distributions, uniform and mod-Gaussian are fully defined by their first two moments and have their mean fixed to the same value, a direct comparison of the second moment suggests itself as a simple but powerful distinguishing method. Thus, we choose the direct variance comparison described in Algorithm 1 due to its simplicity and computational efficiency. Empirically, we found that it was similarly powerful to a Kuiper test, and more than 5x faster in the settings considered.

Other performance considerations. Values of $\mathbf{A}, \mathbf{b}$ are of the order of q and thus large in the considered scenarios. However, high precision is not needed during the enumeration and evaluation, since all precision is drowned out by the large error. Since modern GPUs work best with 16-bit floating point numbers, we scale all values down to a range appropriate for those representations. We also employ `pytorch` compilation to alleviate some of the overhead of python being an interpreted language. This feature compiles, optimizes and fuses tensor operations, giving a 2-3x speed improvement over the default eager execution.

Our implementation can evaluate roughly 5 billion secrets of dimension $n = 512$ through an NVIDIA V100 GPU in about 1 hour during the brute force search. The greedy secret recovery step is fast and runs infrequently. Additional speedups can be had with additional optimizations, such as the use of dedicated CUDA kernels. Additionally, our attack does not yet exploit the recursive search space structure trick from [21].

4 Lattice Reduction

Here, we briefly describe the lattice reduction techniques used to produce the data in our attacks. Classical lattice reduction algorithms, such as LLL, BKZ, BKZ2.0 ([23],[16],[34]) seek to reduce the size of vectors in a lattice. Attacks on LWE such as the uSVP and dual attacks use lattice reduction algorithms to find

the shortest vector, or a *short enough* vector, in a specifically constructed lattice. Our attack leverages lattice reduction methods, but instead of attempting to find a single very short vector, we use reduced lattices as data for a distinguishing attack. For our attack, we apply lattice reduction techniques to the embedded matrix in Equation (3).

Lattice reduction can be viewed as finding an integer-valued transformation matrix $\mathbf{R}$ which, when multiplied by $\mathbf{A}$ modulo q, reduces the Frobenius norm of $\mathbf{A}$. Given LWE samples in the form of a matrix $\mathbf{A}$ and a vector $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, we can use lattice reduction to reduce the entries of the matrix $\mathbf{A}$, yielding $\mathbf{RA}$. Then the corresponding LWE samples are $(\mathbf{RA}, \mathbf{Rb} = \mathbf{RA} \cdot \mathbf{s} + \mathbf{Re})$. Because $\mathbf{R}$ is applied to the noisy vector $\mathbf{b}$ as well, the initial noise in $\mathbf{b}$ is increased by the magnitude of the elements in $\mathbf{R}$.

For this attack, it is not always beneficial to reduce as much as possible, but rather to trade off reduction quality with the magnitude of the error $\mathbf{Re}$ introduced by the reduction via the parameter ω. Intuitively, the stronger the reduction, the higher the error. The error magnitude determines the width of the blue and orange distributions in Figure 2. Section 5 expands on the theory which describes this trade-off.

4.1 Our Reduction Methods

Generating sufficient LWE data. Our attack relies on the ability to distinguish distributions from uniform, and successful distinguishing requires sufficient data. Details are discussed in Section 5 below. An attacker might not have access to enough samples to run our attack in a real world scenario. However, sub-sampling comes to the rescue [25]. From an initial number of m_0 samples, one can select a subset of m_1 samples, create a new $\mathbf{A}$ matrix, embed it in Λ, and perform lattice reduction. This can be done many times, up to $\binom{m_0}{m_1}$ times. This technique "inflates" the number of samples usable for this attack dramatically, at the cost of reduction of many matrices corresponding to different sub-lattices.

The SALSA Picante [25] attack, which proposed this sub-sampling method, eavesdrops $4n$ LWE samples, then sub-samples n samples at a time and applies lattice reduction to $2n \times 2n$ q-ary matrices. Instead, here we sub-sample $m = \text{round}(0.875n)$ from $4n$ LWE samples. We then form q-ary matrices of size $(m + n) \times (m + n)$ and reduce them.

Reduction algorithms. We leverage 3 different algorithms in our lattice reduction step: `BKZ2.0` [16], `flatter` [32], and `polish` [13]. Each algorithm has strengths and weaknesses, and combining them maximizes the strengths of each. `BKZ2.0` provides strong reduction, but is slow for large block size and high dimensional lattices. In contrast, `flatter` is fast and can reduce lattices with $n \geq 512$ and large q, but only has reduction performance analogous to `LLL`. The `polish` algorithm runs at the end of a `flatter` or `BKZ2.0` loop [24] and "polishes" by iteratively orthogonalizing matrices. It provably produces strictly decreasing vector norms when run, is implemented in C, and runs very fast.

10

Measuring performance. We measure lattice reduction performance via ρ, the ratio of the standard deviation of the entries of the reduced lattice to the expected standard deviation of the entries of a random lattice:

$$\rho = \frac{\sigma(\mathbf{A})}{\sigma(\mathbf{A}_{\text{initial}})} = \frac{\sigma(\mathbf{A})}{\frac{q}{\sqrt{12}}}, \tag{5}$$

with $\sigma(\mathbf{A})$ the standard deviation of all entries of $\mathbf{A}$, and $\sigma(\mathbf{A}_{\text{initial}})$ the unreduced, original LWE sample matrix that follows a uniform distribution. When $\rho = 1$, lattice reduction did not succeed. When $\rho \ll 1$, a significant reduction in lattice norms has been achieved. Our attack depends crucially on ρ, since it determines the number of cruel bits in the secret.

Interleaved reduction strategy. Empirically, we found that combining these three algorithms gave the best reduction performance, in terms of both time and quality. `flatter` sometimes gets "stuck" and cannot further reduce a given lattice, particularly for smaller q. In these instances, "interleaving" `BKZ2.0` and `flatter` provided a higher quality reduction while helping `flatter` jump out of these minima (see Table 3 for a comparison of different ways of combining `flatter` and `BKZ2.0`). Our interleaving algorithm starts each reduction run with several rounds of `flatter` and declares a "stall" when `flatter` runs 3 times but produces an average reduction in ρ of less than 0.001. Then, the reduction runs `BKZ2.0`, and when `BKZ2.0` stalls, it switches back to `flatter` and repeats until the reduction converges. `polish` runs after each `BKZ2.0` or `flatter` step.

Table 3. *Comparing performance of 3 preprocessing methods for $n = 512$ and varying $\log_2 q$:* `flatter` *only,* `flatter`*-initialized BKZ (e.g.* `flatter` $\rightarrow$ *BKZ) and Interleaving. Each entry in the table reports ρ/hours.* **Bold** *indicates most reduction ρ with ties broken by shortest time.*

	$\log_2 q$	20	22	24	26	28	30	32	35	38
	Interleave	**0.77/12**	**0.75/12**	0.72/14	**0.69/10**	**0.67/14**	**0.64/24**	**0.57/52**	**0.50/55**	**0.44/56**
$\omega = 4$	flatter→BKZ	0.81/92	0.79/92	0.76/ 92	0.75/114	0.97/72	0.65/109	0.63/55	0.60/63	0.58/60
	flatter only	0.97/72	0.97/72	**0.72/2**	0.7/3	0.68/4	0.68/5	0.58/54	0.58/132	0.46/94
	Interleave	**0.80/11**	**0.77/14**	**0.74/14**	**0.71/14**	**0.68/13**	**0.66/13**	**0.59/53**	**0.53/54**	0.47/57
$\omega = 10$	flatter→BKZ	0.81/93	0.80/93	0.78/93	0.76/114	0.76/92	0.97/72	0.65/50	0.62/55	0.58/58
	flatter only	0.97/72	0.97/72	0.97/72	0.72/2	0.69/3	0.68/7	0.62/74	0.58/130	**0.46/98**

Early termination. To increase the number of reduced LWE matrices produced from a single lattice reduction run, we leverage an early termination strategy. When the standard deviation reduction ratio ρ reaches some threshold τ, we export the reduced matrix, then sub-sample a new $(m+n) \times (m+n)$ q-ary matrix from the initial set of n LWE samples and start again. This better leverages available compute resources.

4.2 Final Datasets

We apply the interleaved reduction strategy described above on the 4 different parameter sets. The number of cruel bits, n_u, is strongly correlated with the

reduction factor ρ: better reduction (lower ρ) yields fewer cruel bits. Because brute-forcing the n_u bits is the bottleneck for our attack, we optimize the lattice reduction configuration to maximize reduction. Reduction is trivially parallelizable, since each matrix can be processed on a different core, but is slower for larger dimensions n and moduli q.

Table 4. *Datasets used in concrete attacks. β_1 and β_2 are block sizes used in lattice reduction. σ_e is the ratio of standard deviation mod q of the magnified error $\mathbf{Re}$ to standard deviation of uniform distribution mod q. n_u is the number of cruel bits. We define any bits with standard deviation greater than $\frac{1}{2}\sigma(A_{initial})$ as cruel bits.*

Inputs			Outputs				
n	$\log_2 q$	β_1, β_2	ρ	Hrs/Matrix	σ_e	n_u	$\rho^2 n$
256	12	$35, 40$	0.769	15	0.952	143	151.3
512	28	$18, 22$	0.677	12	0.692	228	234.6
512	41	$18, 22$	0.413	13.1	0.337	75	87.3
768	35	$18, 22$	0.710	15	0.938	373	387.1

4.3 Quality of Reduction and Our Attack

Experimentally we observe that the number of cruel bits, n_u, is roughly $\rho^2 n$. The data sets presented in Table 4 confirm this observation. Intuitively, this makes sense when assuming column independence and that the standard deviation of the cool bits σ_r is much smaller than that of the cruel bits σ_u:

$$\sigma_{\text{total}}^2 = \frac{1}{n}(n_u \cdot \sigma_u^2 + n_r \cdot \sigma_r^2) \approx \frac{1}{n}(n_u \cdot \sigma_u^2) \qquad (6)$$

During our experiments we usually find $\sigma_r < 0.3\sigma_u$.

5 Statistical tests and analysis

Three statistical tests must be performed during the attack: testing whether a brute force guess of the cruel bits is correct, testing whether cool bits are zero or one (in Algorithm 1), and testing a full secret guess. The first two tests are performed on reduced LWE samples, obtained during the pre-processing phase. The third involves a simple check of residuals on the original LWE sample and will not be discussed here. We elaborate on the statistical properties of these tests, estimate the number of samples needed, and illustrate our results with concrete attack statistics.

5.1 Testing brute-forced guesses of the cruel bits

Suppose a guess of the cruel bits was made, and let $\mathbf{s}^*$ be the secret guess derived by setting all cruel bits to their guessed values and all cool bits to zero. We

are given m reduced LWE samples $(\mathbf{a}, b)$, with the same secret $\mathbf{s}$, error e and reduction factor ρ. We consider the random variable $x = \mathbf{a} \cdot \mathbf{s}^* - b = \mathbf{a} \cdot (\mathbf{s}^* - \mathbf{s}) - e$ mod q, centered to $(-\frac{q}{2}, \frac{q}{2})$. Our intuition from Section 3 is that the standard deviation of x (calculated on m reduced samples) will be lowest when all cruel bits are correctly predicted.

More precisely, let n be the problem dimension and n_u the number of cruel bits. Over reduced LWE samples $(\mathbf{a}, b)$, the n_u first coordinates of $\mathbf{a}$, which correspond to the cruel bits, have standard deviation $\sigma_u = \frac{q}{\sqrt{12}}$ (the standard deviation of the uniform distribution). The standard deviation σ_r of the n_r remaining coordinates of $\mathbf{a}$ (the cool bits) can be derived from the reduction factor ρ and n_u under the assumption that all coordinates of $\mathbf{a}$ are independent after reduction, as follows:

$$\rho^2 \sigma_u^2 = \frac{n_u}{n} \sigma_u^2 + \frac{n_r}{n} \sigma_r^2, \tag{7}$$

and therefore

$$\sigma_r = \sqrt{\frac{\rho^2 n - n_u}{n_r}} \sigma_u. \tag{8}$$

Applying these formulas to the examples in Table 4, we have:

– $\sigma_r = 0.27\sigma_u$ for $n = 256$ and $\log_2 q = 12$ ($\rho = 0.769$ and $n_u = 143$)
– $\sigma_r = 0.15\sigma_u$ for $n = 512$ and $\log_2 q = 28$ ($\rho = 0.677$ and $n_u = 228$)
– $\sigma_r = 0.17\sigma_u$ for $n = 512$ and $\log_2 q = 41$ ($\rho = 0.413$ and $n_u = 75$)
– $\sigma_r = 0.19\sigma_u$ for $n = 768$ and $\log_2 q = 35$ ($\rho = 0.710$ and $n_u = 373$).

When all cruel bits are correctly predicted, and if h_r is the number of ones in the n_r cool bits of the secret (a known value since the Hamming weight of the secret is known), the variance of x is $\sigma_{\mathcal{G}_q}^2(x) = F_q(h_r \sigma_r^2 + \sigma_e^2)$, with $F_q(v)$ the variance modulo q of a centered Gaussian with variance v and a modular operation. We assume independence in the reduced samples and therefore a Gaussian distribution of x before the modulo operation, so the Central Limit Theorem (CLT) holds.

When some cruel bits are incorrect, the residual distribution becomes almost indistinguishable from uniform, because even one incorrect cruel bit significantly increases the Gaussian variance.

Samples needed for cruel bit recovery. We now derive M, the number of reduced samples needed to verify cruel bit guesses, and consequently the amount of pre-processing resources needed for our attack. Since in practice the distribution of residuals is indistinguishable from uniform over $[0, q]$ when the cruel bits are incorrectly chosen, we frame the task of identifying correct cruel bits as distinguishing from uniform. As describe in Section 3.1, we do so by measuring the variance. First, we calculate $\hat{\sigma}^2(x)$ over M reduced samples via

$$\hat{\sigma}^2 = \frac{1}{M} \sum_{i=0}^{M-1} x_i^2. \tag{9}$$

13

We can construct a lower confidence bound ι for the variance under the null hypothesis of uniformity, assuming a Gaussian distribution of $\hat{\sigma}^2$ under the CLT:

$$\iota(\alpha, M) = \sigma^2_{\mathcal{U}_q} + G^{-1}(\alpha) \cdot \sqrt{\frac{\sigma^2_{\mathcal{U}_q^2}}{M}} = \sigma^2_{\mathcal{U}_q} + G^{-1}(\alpha) \cdot \sqrt{\frac{\sigma^2_{\mathcal{U}_q}}{M} \cdot \frac{q^2 - 4}{15}}, \qquad (10)$$

where G^{-1} is the percent point function of $\mathcal{N}(0, 1)$ and $\sigma^2_{\mathcal{U}_q}$ (resp. $\sigma^2_{\mathcal{U}_q^2}$) is the distribution variance of x (resp. x^2) under the null hypothesis of uniformity. α is the type-I error, i.e. the probability of rejecting a true null hypothesis. Since we are brute-forcing a large number of possible solutions, the type-I error, α should be small (i.e. $\alpha <$ the inverse of the number of secret candidates).

Given α and β type-II error (the probability of failing to reject a false null hypothesis), we can estimate the minimum number of samples needed for our attack to succeed with probability $\approx 1 - \beta$ using CLT (see A.2 for more details):

$$M(\alpha, \beta) = \left[\frac{G^{-1}(\alpha)\sigma_{\mathcal{U}_q^2} + G^{-1}(\beta)\sigma_{\mathcal{G}_q^2}}{(\sigma^2_{\mathcal{G}_q} - \sigma^2_{\mathcal{U}_q})} \right]^2 \qquad (11)$$

where $\sigma_{\mathcal{G}_q}$ resp. $\sigma_{\mathcal{G}_q^2}$ is the standard deviation of x resp. x^2 when $x \sim \mathcal{G}_q$ i.e. discrete centered Gaussian mod q. Table 5 gives concrete calculations of M for various h secrets.

Table 5. *Number of samples M needed in the statistical test for type-I error $\alpha = 2^{-128}$ and type-II error $\beta = 10^{-5}$ for LWE settings specified by n, $\log_2 q$, n_u, and secret Hamming weight h. The Worst Case number of samples corresponds to the case where all the 1s in the secret are in the reduced region $h_r = h$, whereas the Average case assumes the average number of 1s in that region, which is the fraction $\frac{n_r}{n}h$*

n	$\log_2 q$	n_u	h	Worst case M $(h_r = h)$	Average case M $(h_r = \frac{n_r}{n}h)$
256	12	143	12	5.67×10^4	1.12×10^4
512	28	228	20	3.29×10^3	1.66×10^3
512	41	75	60	1.98×10^4	1.02×10^4
768	35	373	20	3.13×10^4	9.74×10^3
768	35	373	64	6.21×10^6	1.47×10^5

These results indicate that in the easiest settings (e.g. $n = 512$) a few thousand reduced samples are enough to test cruel bit guesses. For $n = 256$, a few tens of thousands of reduced pairs are needed. As indicated in Section 3.1, the amount of data used can be traded off by running the cool bit estimation and subsequent secret check more often.

5.2 Testing cool bit predictions.

Once the cruel bits have been guessed, we can use Algorithm 1 to recover the cool bits of the secret one bit at a time. As before, we will estimate the standard

deviation of $x = \mathbf{a} \cdot \mathbf{s}^* - b$ over a sample of reduced LWE pairs, but the cruel bits in our guess $\mathbf{s}^*$ are now assumed to be correct. For each cool coordinate, k, we compare two guesses, $\mathbf{s}_0^*$ and $\mathbf{s}_1^*$, which agree with the secret on the $k-1$ first characters, are zero on the $n-k$ characters, and have their k-th bit set to 0 and 1 respectively. If the k-th bit is zero, $x_0 = \mathbf{a} \cdot \mathbf{s}_0^* - b$ should have a lower standard deviation than $x_1 = \mathbf{a} \cdot \mathbf{s}_1^* - b$.

Suppose that all the cruel bits and the $k-1$ first cool coordinates have been correctly guessed, and there are h^* one bits to be discovered in the remaining cool coordinates. Let the null hypothesis be $H_0 : \mathbf{s}_k = 0$ and the alternative hypothesis: $H_1 : \mathbf{s}_k = 1$. The residual $x_0 = \mathbf{a} \cdot \mathbf{s}_0^* - b$ has in both cases variance $\sigma_0^2 = F_q(h^*\sigma_r^2 + \sigma_e^2)$. Under H_0 the variance of $x_1 = \mathbf{a} \cdot \mathbf{s}_1^* - b$ will be $\sigma_{+1}^2 = F_q((h^* + 1)\sigma_r^2 + \sigma_e^2) > \sigma_0^2$. Under H_1 the variance of x_1 is $\sigma_{-1}^2 = F_q((h^* - 1)\sigma_r^2 + \sigma_e^2) < \sigma_0^2$.

In each iteration k of Algorithm 1, our objective is to determine whether the difference in estimated variance, denoted as $\delta(\sigma^2) := \hat{\sigma}^2(x_1) - \hat{\sigma}^2(x_0)$, is more closely aligned with $\sigma_{+1}^2 - \sigma_0^2 > 0$ or $\sigma_{-1}^2 - \sigma_0^2 < 0$. To achieve this, we estimate the variance difference using enough samples. Without the need for optimal threshold tuning, we reject the null hypothesis if the variance difference $\delta(\sigma^2)$ is negative, and accept it otherwise.

Table 6. *Concrete attacks on different secrets: $n =$ dimension, $\log_2 q$ is the modulus size, $h =$ Hamming weight of the secret. h_u indicates the number of cruel bits which are 1. % and cumulative % measure how many secrets with hamming weight h have h_u bits and h_u bits or fewer, respectively. We run all attacks on one GPU to be comparable, but also demonstrate scaling in the toughest setting to 20 GPUs.*

n	$\log_2 q$	h	h_u	% (cum. %)	Samples Used	Time (1 V100 GPU)
256	12	12	4	6.7 (9.5)	200K	28 s
		12	5	14.1 (23.6)		84 s, 241 s
		12	6	21.2 (44.8)		3865 s, 4098 s
		12	7	23.0 (67.8)		23229 s, 26229 s
512	28	12	3	9.5 (13.9)	200K	29 s
		12	4	17.5 (31.4)		70 s
		12	5	22.7 (54.0)		2417 s, 3510 s
512	41	60	7	16.6 (53.6)	1M	376 s, 341 s
		60	8	15.5 (69.1)		1555 s
768	35	10	3	13.1 (19.5)	200K	165 s, 168 s
		10	4	21.7 (41.2)		269 s, 745 s
		12	4	13.5 (22.1)		607 s, 688 s
		12	5	20.5 (42.6)		1291 s (scaled to 20 GPUs)

6 Recovery Results

In this section we provide concrete performance results for our attacks. We run the attack as described in Section 3.1 for different LWE parameter settings and

report results in Table 6. After reduction, the running time depends almost entirely on the amount of enumeration that has to be done on the cruel bits. Enumeration is done in ascending hamming weight on those bits. We extend the secret to the cool bits via the greedy algorithm, every 40M secret candidates, or whenever completing the enumeration on one hamming weight.

All times reported in Table 6 include the compilation time during the first run over a secret, which is of the order of 10 seconds, and loading data, also of the order of 10 seconds. For some secrets, we report multiple timings to illustrate that the process is somewhat noisy, mostly depending on where in the enumeration the secret happens to be. For the settings with higher hamming weights ($h = 60$), we use 200k samples for the brute force attack and 1M for the greedy attack. For the lower hamming weights, we use 10k-30k samples for the brute force attack and 200k for the greedy part.

Existing attacks. For completeness, we compare the performance of our attack to prior work. The hybrid dual meet-in-the-middle (MiTM) attack is most similar to ours [17], but options for comparison are limited, since no concrete performance results are provided in [17]. Some concrete performance results are given for related attacks, but [7] only consider $n \leq 110$, $\log q = 11$, while [30] considers $n \leq 100$, $\log q = 8$. Thus, we choose two routes for comparison. First, we compare against estimated cost of the hybrid dual MiTM estimate from the LWE estimator tool[8], and then we compare against a recent concrete attack implementation for sparse secret LWE proposed in [19]. The latter has code available[†] and claims to be fast. It provides a MITM attack in Python for the $n = 256$, $\log q = 12$ setting and a modified uSVP/BDD attack for larger parameters.

Table 7. *Estimates of Dual Hybrid MiTM performance [8] and concrete attack performance by [19] at parameter settings explored in this work. ROP = estimated number of required operations for attack, Memory = estimated memory required for MiTM hash table, Repeats = # of times attack must run to succeed with probability 0.99999.*

LWE setting	Hybrid MiTM Dual [8]			Ducas et al [19]
(n, q, h)	ROP	Memory	Repeats	Time (sec)
$(256, 12, 12)$	$2^{50.2}$	3.3TB	$2^{13.6}$	200
$(512, 28, 12)$	$2^{48.3}$	2.05TB	$2^{10.4}$	-
$(512, 41, 60)$	$2^{50.4}$	10.8TB	$2^{7.1}$	-
$(768, 35, 12)$	$2^{48.8}$	725GB	$2^{10.6}$	-

Table 7 reports these results. For the estimator [8], we report the estimated number of operations, the memory required, and the number of attack repeats required to achieve a $1 - 10^{-5}$ probability of success (to match our α, β levels in §5). We convert the predicted memory into bytes by multiplying estimator output (number of integers to be stored) by number of bits required to store this integer (32 is suggested). We note that the ROP metric reported by the estimator is a crude estimate for runtime, since it refers to the number of comparisons or

---

[†] https://github.com/lducas/leaky-LWE-Estimator/tree/human-LWE/human-LWE

operations necessary for the attack to run, which does not easily translate to a fixed time metric. For [19], we report time to successful attack, if it succeeds. A '-' in the column indicates the attack ran for ≥ 72 hours (on the same compute hardware as our attack) without finishing.

7 2-power cyclotomic Ring-LWE

Our attack can also be applied in the 2-power cyclotomic Ring-LWE setting. Ring-LWE (RLWE) is a special case of LWE where the LWE samples can be represented more compactly as polynomials in a polynomial ring. Not all instances of Ring-LWE are hard, as was shown in [20]. But 2-power cyclotomic rings are not vulnerable to the attack in [20], so those are the rings standardized for use in Homomorphic Encryption [2].

Consider the 2-power cyclotomic ring defined by the polynomial $x^n + 1$, where $n = 2^k$: $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. Then the RLWE samples corresponding to a polynomial RLWE sample, $(a(x), b(x) = a(x) \cdot s(x) + e(x))$ can be described via a skew-circulant matrix, $\mathbf{A}_{circ}$. If $a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, then:

$$\mathbf{A}_{\text{circ}} = \begin{bmatrix} a_0 & -a_{n-1} & -a_{n-2} & \ldots & -a_1 \\ a_1 & a_0 & -a_{n-1} & \ddots & -a_2 \\ \vdots & a_1 & a_0 & \ddots & -a_{n-1} \\ a_{n-1} & \ddots & \ddots & \ddots & a_0 \end{bmatrix}$$

and $\mathbf{b}_{\text{circ}} = \mathbf{A}_{\text{circ}}\mathbf{s} + \mathbf{e}$ where $\mathbf{s}$ and $\mathbf{e}$ are the vector representations of $s(x)$ and $e(x)$ respectively.

When the matrix has this structure, we can rotate the "cruel bits" around (without redoing any lattice reduction) and increase our chances of recovering the secret, if there are only a small number of cruel bits in one of the rotated cruel regions. This makes the RWLE problem clearly easier than generic LWE in those cases, with respect to our attack.

After reduction, we have reduced samples, $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, where $\mathbf{A} = \mathbf{R}\mathbf{A}_{\text{circ}}$ for some $\mathbf{R} \in \mathbb{Z}^{m \times n}$. We can shift-negate the reduced samples k times to get new samples $(\mathbf{A}^{\leftarrow k}, \mathbf{b}^{\leftarrow k})$ (see A.1). The attack for RLWE consists of performing brute force on n sliding windows $[k, k + n_u \mod n)$ simultaneously (or any other number of windows by using a higher sliding step). The exhaustive search will cover less search space as we only need to check up to $h_u^{(1)} = \min_{0 \leq i < n} h_{n_u, i}(\mathbf{s})$ hamming weight where

$$h_{n_u, i}(\mathbf{s}) = \sum_{j=i}^{i+n_u-1[n]} s_j \tag{12}$$

In this context, $h_{n_u, i}(\mathbf{s})$ represents the Hamming weight of a specific segment of the secret vector $\mathbf{s}$, starting at position i and of length n_u. It's important to
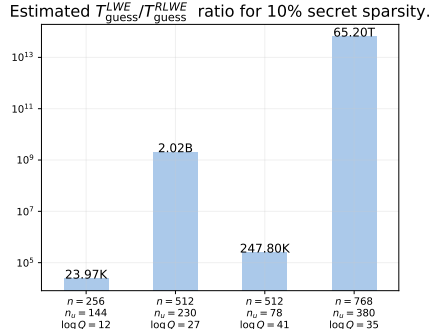
Fig. 3. *Exhaustive search cost ratio for LWE versus RLWE assuming a fixed 10% binary secret sparsity. The values of n_u are obtained experimentally.*

note that when $i = 0$, this notation aligns with our previous use of h_u to denote the Hamming weight of the first n_u elements of $\mathbf{s}$. Similarly, when $n_r = n - n_u$, $h_{n_r,n_u}(\mathbf{s})$ aligns with our previous use of h_r to denote the Hamming weight of the last $n_r = n - n_u$ elements of $\mathbf{s}$.

To evaluate the speed up of this attack compared to the LWE case, we estimate the cost of the brute force component while assuming equal cost of the lattice reduction for LWE and RLWE. Let $c \cdot M(h_r(s), \alpha, \beta)$ represent the cost of checking a secret candidate using $M(h_r(s), \alpha, \beta)$ samples, where c is a constant factor that accounts for the computational resources required for each verification. Let $T_{\text{guess}}^{\text{LWE}}$ be the average time complexity of the exhaustive search for LWE, assuming we sweep through the secret candidates in ascending Hamming weight order:

$$T_{\text{guess}}^{\text{LWE}} = \mathbb{E}_s \left[c \cdot M(h_r(s), \alpha, \beta) \sum_{k=0}^{h_u(s)} \binom{n_u}{k} \right]$$

By using the same number of samples $M = M(h_r(s), \alpha, \beta)$ in the attack for all secret candidates, we have

$$T_{\text{guess}}^{\text{LWE}} = c \cdot M \cdot \sum_{k=0}^{h} \binom{n_u}{k} \mathbb{P}(h_u(s) \geq k)$$

In the RLWE case, we have $\mathbb{P}(h_u^{(1)} \leq \frac{hn_u}{n}) = 1$ so that

$$T_{\text{guess}}^{\text{RLWE}} = n \cdot c \cdot M \cdot \sum_{k=0}^{\lfloor \frac{hn_u}{n} \rfloor} \binom{n_u}{k} \mathbb{P}(h_u^{(1)} \geq k)$$

In Figure 3, we estimate the ratio of the costs above. Note that the distribution of $h_u(s)$ conditionally to a fixed total hamming weight h follows a hyper-geometric distribution when the secret bits are from Bernoulli, while the $h_u^{(1)}$ distribution is computed empirically.

18

We conduct several experiments comparing LWE with RLWE, using identical secrets for comparison. The results of these experiments, for parameters $n = 256$ and $n = 512$, are presented in Table 8 and Table 9. Each table row corresponds to a distinct secret s, and the columns detail the actual and estimated time costs for the LWE and RLWE attacks, along with the count of cruel bits in the secret. In the tables, Actual Time is the empirically measured duration of the attack, while the Estimated Time is estimated using an approximation $T_{\text{guess}}^{\text{LWE}} = b + a \sum_{k=0}^{h_u(s)} \binom{n_u}{k}$. This estimate provides an idea of the time required to attack a secret when h_u is very large, without actually running the computation.

Table 8. *Attack times for parameters: $n = 256, h = 12, \log q = 12, n_u = 145$ run on 20 randomly sampled secrets. Actual time is the empirical time of the attack. Estimated time is computed via the formula: $T_{guess}^{LWE} = b + a \sum_{k=0}^{h_u(s)} \binom{n_u}{k}$ and $T_{guess}^{RLWE} = n(b + a \sum_{k=0}^{h_u^{(1)}} \binom{n_u}{k})$ for RLWE. $h_u(s)$, as defined in Equation (12), is the number of the cruel bits of the secret, and $h_u^{(1)}$ is the minimum hamming weight over all windows of length n_u. We use 56k samples for Cruel bit recovery and 200k for Algorithm 1.*

	LWE			RLWE		
Secret	Estimated time (sec)	Actual time (sec)	$h_u(s)$	Estimated time	Actual time	$h_u^{(1)}(s)$
1	2.33×10^1	2.21×10^1	3	5.86×10^3	4.83×10^3	2
2	4.60×10^2	3.59×10^2	5	1.18×10^5	1.21×10^5	5
3	4.60×10^2	5.24×10^2	5	1.18×10^5	1.73×10^5	5
4	4.60×10^2	4.12×10^2	5	9.77×10^3	1.02×10^4	4
5	4.60×10^2	2.61×10^2	5	5.97×10^3	4.58×10^3	3
6	4.60×10^2	3.61×10^2	5	9.77×10^3	1.19×10^4	4
7	1.04×10^4	–	6	5.97×10^3	6.00×10^3	3
8	2.09×10^5	–	7	1.18×10^5	1.76×10^5	5
9	2.09×10^5	–	7	1.18×10^5	1.48×10^5	5
10	2.09×10^5	–	7	9.77×10^3	1.20×10^4	4
11	2.09×10^5	–	7	9.77×10^3	1.15×10^4	4
12	2.09×10^5	–	7	5.97×10^3	5.71×10^3	3
13	2.09×10^5	–	7	1.18×10^5	5.38×10^4	5
14	3.66×10^6	–	8	9.77×10^3	1.06×10^4	4
15	3.66×10^6	–	8	9.77×10^3	1.26×10^4	4
16	3.66×10^6	–	8	9.77×10^3	1.19×10^4	4
17	5.65×10^7	–	9	5.97×10^3	4.12×10^3	3
18	5.65×10^7	–	9	1.18×10^5	1.33×10^5	5
19	5.65×10^7	–	9	5.97×10^3	6.02×10^3	3
20	7.81×10^8	–	10	5.86×10^3	4.74×10^3	2
Avg	4.81×10^7	9.63×10^1	6.85×10^0	4.08×10^4	4.61×10^4	3.85×10^0

Note that for RLWE, the Time is multiplied by n to account for the time for running the n brute force attacks, although they can be run simultaneously. The final row provides the average values across all secrets. As can be seen, the RLWE attack is significantly faster on average, with the ratio of average times

being $\sim 10^3$ for $n = 256$ and 5×10^3 for $n = 512$, but converge to the theoretical values in Figure 3 when estimated on more secret samples. The RLWE advantage can also be seen in the values of $h_u^{(1)}(s)$ which are much smaller than those of $h_u(s)$.

Table 9. *Attack times (in seconds) and estimated costs for $n = 512, n_u = 75, h = 60, \log q = 41$ for random 20 secrets sorted by number of secret cruel bits in LWE. We use $40K$ samples for cruel bit recovery and $1.5M$ for Algorithm 1*

Secret	LWE			RLWE		
	Estimated time (sec)	Actual time (sec)	$h_u(s)$	Estimated time	Actual time	$h_u^{(1)}(s)$
1	6.88×10^2	5.95×10^2	6	1.82×10^5	7.74×10^4	4
2	6.88×10^2	2.82×10^3	6	1.95×10^5	6.31×10^4	5
3	3.98×10^3	3.93×10^3	7	1.82×10^5	7.68×10^4	4
4	3.45×10^4	–	8	3.52×10^5	1.08×10^5	6
5	2.82×10^5	–	9	1.82×10^5	4.62×10^4	4
6	2.82×10^5	–	9	1.82×10^5	5.01×10^4	4
7	2.82×10^5	–	9	1.95×10^5	8.72×10^4	5
8	2.82×10^5	–	9	1.95×10^5	1.15×10^6	5
9	2.06×10^6	–	10	3.52×10^5	–	6
10	2.06×10^6	–	10	1.82×10^5	4.64×10^4	4
11	2.06×10^6	–	10	1.81×10^5	–	3
12	2.06×10^6	–	10	1.81×10^5	–	2
13	1.36×10^7	–	11	1.82×10^5	6.21×10^4	4
14	1.36×10^7	–	11	1.95×10^5	9.58×10^4	5
15	1.36×10^7	–	11	1.82×10^5	–	4
16	1.36×10^7	–	11	1.82×10^5	3.56×10^4	4
17	8.06×10^7	–	12	1.81×10^5	4.43×10^4	3
18	4.36×10^8	–	13	1.82×10^5	–	4
19	9.88×10^9	–	15	1.81×10^5	4.46×10^4	3
20	9.88×10^9	–	15	1.81×10^5	4.67×10^4	2
Avg	1.02×10^9		10.1	2.01×10^5		4.05

8 Discussion and Future Work

We have presented an attack on Learning With Errors in the setting of sparse binary secrets. Our key insight is the reduction of the LWE problem to one of smaller dimension via lattice reduction and sufficient subsampling. We discuss the attack empirically and theoretically and highlight actual successful attacks for dimension up to $n = 768$. We leave several important tasks as future work.

Improving upon combinatorial scaling in reduced problem. We use brute force to solve the smaller LWE problem (e.g. in the cruel region). This is the crudest way to solve this problem, albeit perfectly parallelizable. The guessing

parts of hybrid attack strategies of related work could speed up enumeration. One related way in which we already exploit this is in the 2-cyclotomic RLWE setting. But of course, the rotation (or more generally, permutation) of the cruel bit region can of course be done on the LWE settings as well. We did not elaborate on that (hybrid) approach because it requires re-reduction of lattices.

Balancing reduction and secret recovery costs. Currently, our attack runs lattice reduction until it stalls for a given n/q setting, then runs secret recovery for the highest h that can be achieved in reasonable time. In the future, our attack should balance the reduction and enumeration costs given the setting of concern, also depending on how hybrid attacks will be employed.

Recovering the cool bits. As Section 5.2 demonstrates, cool bit recovery is somewhat sample-inefficient, because Algorithm 1 must distinguish between $F_q(k\sigma_r^2 + \sigma_e^2)$ and $F_q((k+1)\sigma_r^2 + \sigma_e^2)$. This difference tends to be small when k is large, and σ_e is large, compared to σ_r, necessitating more samples. This suggests an additional consideration for the lattice reduction phase: balancing the amount of reduction achieved and the noise added in the process. When reduction factor ρ is smaller, σ_r is smaller, resulting in fewer cruel bits and easier recovery. On the other hand, a smaller ρ means a larger σ_e, necessitating more data during the attack. We leave the quantitative analysis of this relationship as future work.

References

1. Ajtai, M.: Generating hard instances of lattice problems. In: Proc. of the 28th annual ACM symposium on Theory of Computing (1996)
2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic encryption standard. In: Protecting Privacy through Homomorphic Encryption. Springer (2021), https://eprint.iacr.org/2019/939
3. Albrecht, M., Ducas, L.: Lattice attacks on ntru and lwe: a history of refinements. Cryptology ePrint Archive (2021)
4. Albrecht, M.R.: On Dual Lattice Attacks Against Small-Secret LWE and Parameter Choices in HElib and SEAL. In: Coron, J.S., Nielsen, J.B. (eds.) Proc. of EUROCRYPT (2017), https://eprint.iacr.org/2017/047
5. Albrecht, M.R., Curtis, B.R., Deo, A., Davidson, A., Player, R., Postlethwaite, E.W., Virdia, F., Wunderer, T.: Estimate all the LWE, NTRU schemes! Cryptology ePrint Archive, Paper 2018/331 (2018), https://eprint.iacr.org/2018/331, https://eprint.iacr.org/2018/331
6. Albrecht, M.R., Fitzpatrick, R., Göpfert, F.: On the efficacy of solving lwe by reduction to unique-svp. Cryptology ePrint Archive, Paper 2013/602 (2013), https://eprint.iacr.org/2013/602, https://eprint.iacr.org/2013/602
7. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to lwe. In: Proc. of ASIACRYPT. pp. 297–322. Springer (2017), https://eprint.iacr.org/2017/815.pdf
8. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology (2015), https://eprint.iacr.org/2015/046.
9. Bai, S., Galbraith, S.D.: Lattice Decoding Attacks on Binary LWE. In: Information Security and Privacy (2014)

10. Bi, L., Lu, X., Luo, J., Wang, K., Zhang, Z.: Hybrid dual attack on lwe with arbitrary secrets. Cryptology ePrint Archive, Paper 2021/152 (2021), https://eprint.iacr.org/2021/152, https://eprint.iacr.org/2021/152

11. Buchmann, J., Büscher, N., Göpfert, F., Katzenbeisser, S., Krämer, J., Micciancio, D., Siim, S., van Vredendaal, C., Walter, M.: Creating Cryptographic Challenges Using Multi-Party Computation: The LWE Challenge. In: Proc. of the 3rd ACM International Workshop on ASIA Public-Key Cryptography (2016)

12. Buchmann, J., Göpfert, F., Player, R., Wunderer, T.: On the hardness of lwe with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. Cryptology ePrint Archive, Paper 2016/089 (2016), https://eprint.iacr.org/2016/089, https://eprint.iacr.org/2016/089

13. Charton, F., Lauter, K., Li, C., Tygert, M.: An efficient algorithm for integer lattice reduction. arXiv preprint arXiv:2303.02226 (2023)

14. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Proc. of EUROCRYPT (2018), https://eprint.iacr.org/2018/067

15. Chen, L., Moody, D., Liu, Y.K., et al.: PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates. US Department of Commerce, NIST (2022), https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

16. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better Lattice Security Estimates. In: Proc. of ASIACRYPT 2011 (2011)

17. Cheon, J.H., Hhan, M., Hong, S., Son, Y.: A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE. IEEE Access (2019)

18. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security (2017), https://eprint.iacr.org/2016/421.pdf

19. Ducas, L., Postlethwaite, E., Sotakova, J.: SALSA Verde vs. The Actual State of the Art (2023), https://crypto.iacr.org/2023/rump/crypto2023rump-paper13.pdf

20. Elias, Y., Lauter, K.E., Ozman, E., Stange, K.E.: Provably weak instances of ring-lwe. In: Advances in Cryptology – CRYPTO 2015. pp. 63—-92 (2015)

21. Espitau, T., Joux, A., Kharchenko, N.: On a hybrid approach to solve small secret lwe. Cryptology ePrint Archive, Paper 2020/515 (2020), https://eprint.iacr.org/2020/515, https://eprint.iacr.org/2020/515

22. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against ntru. In: Proc. of CRYPTO. Springer (2007)

23. Lenstra, H.j., Lenstra, A., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 515–534 (1982)

24. Li, C., Wenger, E., Allen-Zhu, Z., Charton, F., Lauter, K.: SALSA VERDE: a machine learning attack on Learning With Errors with sparse small secrets. In: Proc. of NeurIPS (2023)

25. Li, C.Y., Sotáková, J., Wenger, E., Malhou, M., Garcelon, E., Charton, F., Lauter, K.: Salsa Picante: A Machine Learning Attack on LWE with Binary Secrets. In: Proc. of ACM CCS (2023)

26. Lindner, R., Peikert, C.: Better Key Sizes (and Attacks) for LWE-Based Encryption. In: Topics in Cryptology – CT-RSA 2011 (2011)

27. Liu, M., Nguyen, P.Q.: Solving BDD by Enumeration: An Update. In: Dawson, E. (ed.) Topics in Cryptology – CT-RSA 2013 (2013)

28. Micciancio, D., Regev, O.: Lattice-based cryptography. Post-Quantum Cryptography pp. 147–191 (2009)

29. Paszke, A., Gross, S., Massa, F., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Proc. of NeurIPS (2019)
30. Postlethwaite, E.W., Virdia, F.: On the success probability of solving unique svp via bkz. In: IACR International Conference on Public-Key Cryptography. pp. 68–98. Springer (2021)
31. Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In: Proc. of the Thirty-Seventh Annual ACM Symposium on Theory of Computing (2005), https://dblp.org/rec/journals/corr/cs-DS-0304005.bib
32. Ryan, K., Heninger, N.: Fast practical lattice reduction through iterated compression. Cryptology ePrint Archive (2023)
33. Son, Y., Cheon, J.H.: Revisiting the hybrid attack on sparse and ternary secret lwe. Cryptology ePrint Archive, Paper 2019/1019 (2019), https://eprint.iacr.org/2019/1019, https://eprint.iacr.org/2019/1019
34. development team, T.F.: fplll, a lattice reduction library, Version: 5.4.4 (2023), https://github.com/fplll/fplll, available at https://github.com/fplll/fplll

A Appendix

A.1 Rotating Reduced Short 2-cyclotomic RLWE Vectors

Consider the R_q-endomorphism ϕ defined by $\phi : a \mapsto xa$. We establish its counterpart in the canonical embedding as:

$$\Phi(A) = AX, \quad \forall A \in \mathbb{Z}^{m \times n},$$

where X is defined as $X = \text{Circ}(x_{\text{vec}})^\top$, with Circ denoting the skew-circulant and x being the degree 1 polynomial whose vector representation is given by $x_{\text{vec}} = (0, 1, 0, \ldots, 0) \in \mathbb{Z}^n$. We further introduce the notation $\Phi_k(A)$ to denote the k-fold composition of Φ, i.e., $\Phi^k(A) = \Phi \circ \Phi \circ \cdots \circ \Phi$.

if $A_{\text{circ}} = \text{Circ}(a)$, then A_{circ} and X commute: $XA_{\text{circ}} = A_{\text{circ}}X$. This is due to the associativity in the ring R_q.

This allows us to have new samples $(A^{\to k}, b^{\to k})$ by shift-negating the reduced samples k times:

$$A^{\to k} := AX^k = \Phi_k(A)$$

and

$$
\begin{aligned}
A^{\to k}s &= RA_{\text{circ}}X^k s \\
&= RX^k A_{\text{circ}} s \\
&= RX^k (A_{\text{circ}}s + e) - RX^k e \\
&= \Phi_k(R)b_{\text{circ}} - \Phi_k(R)(e)
\end{aligned}
$$

So by defining $b^{\to k} := R^{\to k}b_{\text{circ}}$, $(A^{\to k}, b^{\to k})$ are LWE samples with secret s.

We illustrate empirically how subselecting samples from different indicies of the circulant matrix affects h_u in Figure 4. When we subsample elements when $k = 0$ (e.g. no shifting), $h_u = 16$ for a $n = 256 \log q = 12$, $h = 20$ problem. However, subsampling elements at $k = 121$ yields $h_u = 4$, a much easier secret.
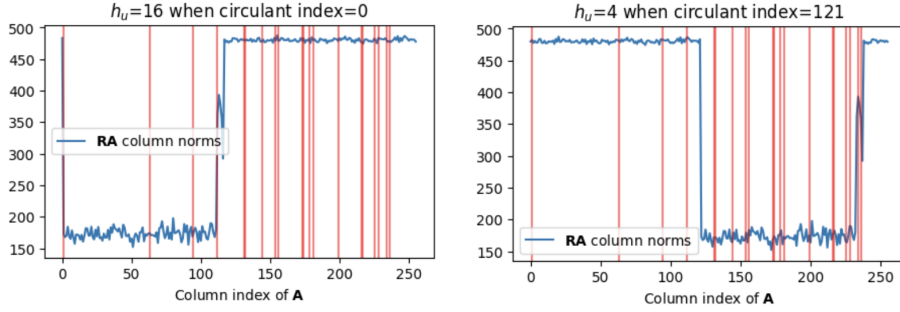
Fig. 4. *Effect of shifting the RLWE circulant matrix on number of h_u secret bits, for $n = 256$, $\log_2 = 12$, $n_u = 143$, $h = 20$.*

A.2 Minimum Sample Requirement for Hypothesis Testing

We examine two centered discrete distributions, namely $\mathcal{U}_q := Uniform(\mathbb{Z}_q)$ and $\mathcal{G}_q := Discrete\ Gaussian(0, \sigma_{\mathcal{G}}^2) \mod q$ where $\sigma_{\mathcal{G}}^2 = h_r \sigma_r^2 + \sigma_e^2$ and $\sigma_{\mathcal{G}_q}^2 = F_q(\sigma_{\mathcal{G}}^2)$.

The variance of x^2 when x is uniform is $\sigma_{\mathcal{U}_q^2}^2 = \sigma_{\mathcal{U}_q}^2 \frac{q^2 - 4}{15}$. For the discrete Gaussian $\mod q$, the variance of x^2: $\sigma_{\mathcal{G}_q^2}^2 = \mathbb{E}_{\mathcal{G}_q}(x^4) - \mathbb{E}_{\mathcal{G}_q}(x^2)^2$ is approximated. Let x be a sample drawn from either of the two aforementioned distributions. We also consider M samples x_i drawn from the same distribution as x. To differentiate between the two distributions, we conduct a statistical test with the following null and alternative hypotheses:

$$\textbf{H0: } \sigma^2(x) = \sigma_{\mathcal{U}_q}^2 \quad \textbf{H1: } \sigma^2(x) = \sigma_{\mathcal{G}_q}^2$$

We consider the unbiased variance estimator for a known 0 mean: $\hat{\sigma}_M^2 = \frac{1}{M} \sum_{i=0}^{M-1} x_i^2$.

The threshold $\iota(\alpha, M)$ is set so that the type-1 error: $\alpha = \mathbb{P}_{\mathcal{U}_q}(\hat{\sigma}_M^2 < \iota(\alpha, M)) = \mathbb{P}(\hat{\sigma}_M^2 < \iota(\alpha, M)|x \sim \mathcal{U}_q)$ is given. Using Central Limit Theorem and solving for $\iota(M, \alpha)$

$$\alpha = \mathbb{P}(\hat{\sigma}_M^2 < \iota | x \sim \mathcal{U}_q) = \mathbb{P}\left(\sqrt{\frac{M}{\sigma_{\mathcal{U}_q^2}^2}}(\hat{\sigma}_M^2 - \sigma_{\mathcal{U}_q}^2) < \sqrt{\frac{M}{\sigma_{\mathcal{U}_q^2}^2}}(\iota - \sigma_{\mathcal{U}_q}^2)|x \sim \mathcal{U}_q\right)$$

$$\approx \mathbb{P}\left(\mathcal{G}(0,1) < \sqrt{\frac{M}{\sigma_{\mathcal{U}_q^2}^2}}(\iota - \sigma_{\mathcal{U}_q}^2)\right) = G\left(\sqrt{\frac{M}{\sigma_{\mathcal{U}_q^2}^2}}(\iota - \sigma_{\mathcal{U}_q}^2)\right)$$

Which results in the equation:

$$\iota(M, \alpha) = \sigma_{\mathcal{U}_q}^2 + G^{-1}(\alpha)\frac{\sigma_{\mathcal{U}_q^2}}{\sqrt{M}} \tag{13}$$

Doing the same for the type-2 error given $\iota(M, \alpha)$ and solving for M:

$$\beta = \mathbb{P}(\hat{\sigma}_M^2 \geq \iota(M, \alpha)|x \sim \mathcal{G}_q) \approx G\left(\sqrt{\frac{M}{\sigma_{\mathcal{G}_q^2}^2}}(\sigma_{\mathcal{G}_q}^2 - \iota(M, \alpha))\right)$$

By substituting $\iota(M, \alpha)$ by its expression in Equation (13), we have:

$$M = \left[\frac{G^{-1}(\alpha)\sigma_{\mathcal{U}_q^2} + G^{-1}(\beta)\sigma_{\mathcal{G}_q^2}}{(\sigma_{\mathcal{G}_q}^2 - \sigma_{\mathcal{U}_q}^2)}\right]^2$$

A.3   Our observation and the q-ary lattice z-shape

To avoid confusion, we comment briefly on a similar looking phenomenon. Prior work has observed a "z-shape" in q-ary lattices and leveraged it in the so-called hybrid attack [22,3]. This classic z-shape is exhibited by the Gram-Schmidt orthogonalized rows of a q-ary lattice before and after lattice reduction (see left plot of Figure 5). This is distinct from the behavior we observe in the columns of **A** after reduction (right figure of Figure 5).
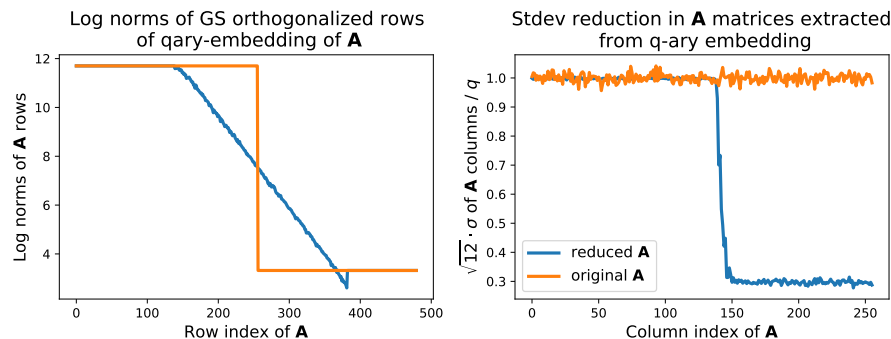

Fig. 5. *The log-norms of Gram-Schmidt orthogonalized rows of q-ary embedded* **A** *(left) and the stdev of columns of* **A** *after it is extracted from the q-ary embedding. Both results are for* $n = 256$, $\log_2 q = 12$ *LWE problems, using the same* **A** *matrix.*