Accelerating the Primal Hybrid Attack against Sparse LWE using GPUs

Ludo N. Pulles[1] and Paul Vié[2]

[1] CWI, Cryptology Group, Amsterdam, the Netherlands
[2] Télécom Paris, Paris, France

Abstract. Although the lattice-estimator predicts that Learning with Errors instances having small and very sparse secrets can be broken by hybrid attacks with modest computational resources, no efficient open-source implementation of these attacks exists. This work implements the so-called Guess + Verify attack (G+V) analysed by Albrecht et al. (SAC'19), containing three improvements: (1) cuBLASter, a GPU-based implementation of the lattice basis reduction software BLASter by Ducas et al. (ASIACRYPT'25); (2) a dimension reduction technique for the BDD instance; and (3) a batched variant of Babai's Nearest Plane algorithm.
On bases of dimension 512 and above, cuBLASter outperforms BLASter. We also integrate the GPU implementation of the General Sieve Kernel by Ducas et al. (EUROCRYPT'21) into cuBLASter's BKZ framework. Running G+V on the benchmark instances by Wenger et al. (IEEE SP'25), we show that G+V achieves significantly higher success rates than the Cool&Cruel attack (C+C) by Nolte et al. (AFRICACRYPT'24) on almost all instances, and G+V's average CPU and GPU utilization is substantially lower than the minimum reported by C+C.

Keywords: Bounded Distance Decoding · Learning with Errors · Hybrid Attack · Lattice Reduction · Graphics Processing Unit

1 Introduction

Lattices are shown to be useful in constructing efficient post-quantum KEMs and signature schemes [SAB$^+$22,LDK$^+$22,PFH$^+$22]. The security of these schemes is based on the hardness of the Bounded Distance Decoding (BDD) problem in lattices constructed using Learning with Errors (LWE) [Reg09] or NTRU [HPS98], and the hardness of BDD is well-understood in these cases from a theoretical viewpoint [APS15,YD17]. However, many implementations for Fully Homomorphic Encryption [BGV12,Bra12,FV12,CKKS17] use LWE with a small and sparse secret inside bootstrapping [CH18,CHK$^+$18,HS21] for efficiency reasons. Because the search space is smaller in so-called sparse LWE, there are possibly more efficient attacks and the efficiency of these attacks is not entirely clear [ACC$^+$18,BCC$^+$24].

The currently best known attack on sparse secret LWE is a hybrid of lattice reduction [SE94] and a combinatorial part [How07,BGPW16,ACW19]. In theory, it is better to use Meet in the Middle (MitM) for the combinatorial part instead of exhaustive search, but MitM requires a large amount of memory, and the hybrid MitM uses multiple heuristics, which resulted in some incorrect security estimations [Wun19,Ngu21]. Such issues possibly could have been avoided if the algorithm was benchmarked and validated on small instances. Further improvements to this MitM hybrid have been proposed in the literature [HKLS22,BLLW22], which take advantage of representation techniques [May21]. This makes it even more important to know how effective these attacks are in practice.

Lattice attacks generally require reducing a lattice basis using *block Korkine–Zolotarev* (BKZ) [SE94,CN11]. When running BKZ using a small block size β parameter, this algorithm repeatedly solves the Shortest Vector Problem (SVP) in dimension β efficiently using enumeration [GNR10], and `fplll` provides a good implementation for this. For larger block sizes of roughly $\beta \geq 60$, it is shown [Duc18] that SVP in dimension β can be solved more efficiently using lattice sieving [BDGL16]. The *General Sieve Kernel* (`G6K`) [ADH$^+$19] provides an open source implementation for state of the art lattice sieving, and can even run on graphics cards (GPUs) [DSvW21].

Recently, two attacks on sparse LWE have been published: the Salsa attack [LWAZ$^+$23,SWL$^+$25], which is based on machine learning, and the Cool & Cruel attack [NMW$^+$24]. The authors of both works provide an implementation to run these attacks on a large computing cluster utilizing many CPUs and GPUs.

The implementations of these two attacks are compared in [WSM$^+$25] with the generic uSVP attack [AGVW17], which embeds BDD into a lattice and then extracts the secret using BKZ. A pure MitM approach [HGSW03] is shown to outperform Salsa Verde on the first instance of [LWAZ$^+$23, Table 15] in a rump session [DPS23]. Moreover, Cool & Cruel outperforms Salsa in benchmarks [WSM$^+$25]. Recently, [KKN$^+$25] argues that Cool & Cruel can be seen as some kind of dual hybrid attack, and [KKN$^+$25] shows that Cool & Cruel performs comparably in wall time to a CPU-only implementation of a simple primal hybrid attack, called Drop + Solve [ACW19]. Still, some of the above-mentioned hybrid attacks that are known to be fastest in theory need to be implemented to provide a complete, practical baseline of all hybrid attacks on sparse LWE. In addition, the implementations of Cool & Cruel and Salsa make extensive use of parallelization in CPUs and GPUs, and while the open-source implementations use CPU-based parallelism using the recent lattice reduction improvement called `flatter` [RH23], these hybrid attacks could still be accelerated using GPUs to provide a comparison to Salsa and Cool & Cruel that one may consider more fair.

1.1 Contributions

In this work, we accelerate the BLASter lattice reduction library [DPS25] using GPUs, called cuBLASter, which provides a fast library for LLL, DeepLLL and

BKZ (for small block sizes). Moreover, this implementation is able to call the GPU-version of `G6K` to run BKZ for block sizes of at least 60.

In addition, we implement the Guess + Verify attack [ACW19], having three notable improvements.

1. This implementation uses cuBLASter for lattice basis reduction.
2. We reduce the BDD distinguishing problem to a projected sublattice, where the dimension reduction improves the BDD solver, which is based on Babai's Nearest Plane (NP) algorithm [Bab86].
3. We execute Babai's NP algorithm in batches [DPS25, App. A.1], and make use of efficient matrix multiplication implementations.

Implementation. The lattice reduction library cuBLASter and the implementation of the Guess + Verify attack are both open source and available on GitHub, respectively:

- `https://github.com/ludopulles/cuBLASter`
- `https://github.com/ludopulles/GPUPrimalHybrid`

Acknowledgments. We would like to thank Alexandr Karenin and Fernando Virdia for running some of our experiments. The first author was supported by ERC Starting Grant 947821 (ARTICULATE). Part of this work was done while the second author was an intern at Centrum Wiskunde & Informatica.

2 Preliminaries

This section explains the sparse LWE problem, and the primal hybrid attack. Notation will be reused from [KKN+25], in particular Sections 2 and 5.

We generally use column vectors. Vectors and matrices have lowercase and uppercase names respectively, and are boldfaced. The identity matrix of order n is denoted by id_n. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and numbers $0 \le a < b \le m$, $0 \le c < d \le n$, by counting from zero, $\boldsymbol{A}_{[a:b,c:d]}$ denotes the submatrix consisting of rows $a, a+1, \ldots, b-1$ and columns $c, c+1, \ldots, d-1$, which matches the indexing used by NumPy.

The integers modulo q is denoted by $\mathbb{Z}/q\mathbb{Z}$. The *uniform distribution* on the interval $[a, b]$ is denoted by $\mathrm{U}(a, b)$. The *support* of a vector $\boldsymbol{x} \in \mathbb{R}^n$, denoted by $\mathrm{Supp}(\boldsymbol{x})$, consists of the indices $i \in \{1, \ldots, n\}$ such that $\boldsymbol{x}_i \neq 0$. The *normal distribution* with mean μ and variance σ^2 is denoted $\mathcal{N}(\mu, \sigma^2)$; for vectors, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the multivariate normal with covariance matrix $\boldsymbol{\Sigma}$. The *chi-squared distribution* with d degrees of freedom is denoted χ^2_d; if $X \sim \mathcal{N}(0, 1)^d$, then $\|X\|^2 \sim \chi^2_d$. For a distribution D with cumulative distribution function F_D, we denote by $q_D(p)$ the p-quantile: $q_D(p) = \inf\{x : F_D(x) \ge p\}$. We write $\mathrm{mod}^{\pm} p$ to denote reduction of integers modulo p to the interval $[-\lfloor p/2 \rfloor, \lceil p/2 \rceil]$.

2.1 Lattices

A lattice $\varLambda \subseteq \mathbb{R}^d$ is a discrete subgroup of $\mathbb{R}^d$. For any matrix $\boldsymbol{B} \subseteq \mathbb{R}^{d\times n}$, let $\mathcal{L}(\boldsymbol{B}) = \{\boldsymbol{x} \in \mathbb{R}^d \,|\, \exists \boldsymbol{c} \in \mathbb{Z}^n \colon\ \boldsymbol{x} = \boldsymbol{B}\boldsymbol{c}\}$. The matrix $\boldsymbol{B} \subseteq \mathbb{R}^{d\times n}$ is *a basis* for $\varLambda$ if all columns of $\boldsymbol{B}$ are $\mathbb{R}$-linearly independent and $\varLambda = \mathcal{L}(\boldsymbol{B})$ holds. Any lattice has a basis. Given a basis $\boldsymbol{B} \subseteq \mathbb{R}^{d\times n}$ for a lattice $\varLambda \subseteq \mathbb{R}^d$, we say $\varLambda$ is of *rank* n, and say $\varLambda$ is *full-rank* if $n = d$.

The *determinant* of a lattice $\varLambda$ is given by $\det(\varLambda) = \sqrt{\det(\boldsymbol{B}^\intercal \boldsymbol{B})}$, where $\boldsymbol{B}$ is any basis of $\varLambda$. Note this value does not depend on the choice of basis $\boldsymbol{B}$. The *first minimum* of $\varLambda$ is the norm of the shortest nonzero vector of $\varLambda$, and is denoted by $\lambda_1(\varLambda) = \min\{\|\boldsymbol{x}\| \,:\, \boldsymbol{x} \in \varLambda \setminus \{0\}\}$.

Given a basis $\boldsymbol{B} = [\boldsymbol{b}_0 \cdots \boldsymbol{b}_{n-1}]$ of $\varLambda$, we define two families of projection operators on $\mathbb{R}^d$. For $k \in \{0, 1, \ldots, n\}$, denote by π_k the orthogonal projection onto $\mathrm{Span}_{\mathbb{R}}(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{k-1})$, and $\pi_k^\perp \colon \boldsymbol{x} \mapsto \boldsymbol{x} - \pi_k(\boldsymbol{x})$ denotes the projection orthogonally away from that space. If needed for clarity, we make the dependence on $\boldsymbol{B}$ explicit: $\pi_{\boldsymbol{B},k}$. Note $\pi_0(\boldsymbol{x}) = \boldsymbol{0}$ and $\pi_n(\boldsymbol{x}) = \boldsymbol{x}$ holds for all $\boldsymbol{x} \in \mathrm{Span}_{\mathbb{R}}(\varLambda)$.

For $0 \le \ell < r \le n$ and a basis $\boldsymbol{B} = [\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}] \in \mathbb{R}^{d\times n}$, we write

$$\boldsymbol{B}_{[\ell,r)} = \left[\pi_\ell^\perp(\boldsymbol{b}_\ell), \ldots, \pi_\ell^\perp(\boldsymbol{b}_{r-1})\right],$$

which is a basis for a *projected sublattice* of $\mathcal{L}(\boldsymbol{B})$.

The *Gaussian Heuristic* states for any set S and full-rank random lattice $\varLambda$, one expects $|S \cap \varLambda| \approx \mathrm{Vol}_d(S) \,/\, \det \varLambda$. In particular, let us define

$$\mathrm{GH}(d) = \mathrm{Vol}_d\left(\mathcal{B}^d\right)^{-1/d} = \Gamma(d/2+1)^{1/d} / \sqrt{\pi},$$

where Γ denotes the gamma function. The *Gaussian Heuristic* predicts a random lattice satisfies $\lambda_1(\varLambda) \approx \det(\varLambda)^{1/d}\mathrm{GH}(d)$.

The *QR decomposition* of a full-rank basis $\boldsymbol{B} \in \mathbb{R}^{d\times n}$ is the unique pair $(\boldsymbol{Q}, \boldsymbol{R}) \in \mathbb{R}^{d\times n} \times \mathbb{R}^{n\times n}$ such that $\boldsymbol{R}$ is an upper-triangular matrix with positive diagonal, and $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$ and $\boldsymbol{Q}^\intercal \boldsymbol{Q} = \mathrm{id}_n$ hold.

The *fundamental domain* of a basis $\boldsymbol{B}$ is given by

$$\mathcal{P}(\boldsymbol{B}) = \left\{\boldsymbol{y} \in \mathbb{R}^d \,\middle|\, \exists \boldsymbol{c} \in \left[-\tfrac{1}{2}, \tfrac{1}{2}\right)^n \colon \boldsymbol{y} = \boldsymbol{B}\boldsymbol{c}\right\}.$$

The ith *Gram–Schmidt* vector is given by $\mathbf{b}_i^* = \pi_i(\boldsymbol{b}_i)$, and the *Gram–Schmidt orthogonalization* of $\boldsymbol{B}$ is $\mathbf{B}^* = \left[\mathbf{b}_0^*, \ldots, \mathbf{b}_{n-1}^*\right]$. The *Babai domain* of $\boldsymbol{B}$ is $\mathcal{P}(\mathbf{B}^*)$. Babai's *Nearest Plane Algorithm* (NP) [Bab86] is an algorithm that upon input $\boldsymbol{B}$ and $\boldsymbol{t} \in \boldsymbol{B}$ outputs the unique pair $(\boldsymbol{c}, \boldsymbol{e}) \in \mathbb{Z}^n \times \mathbb{R}^d$ satisfying

$$\boldsymbol{t} = \boldsymbol{B}\boldsymbol{c} + \boldsymbol{e}, \quad \text{and} \quad \pi_n(\boldsymbol{e}) \in \mathcal{P}(\mathbf{B}^*).$$

2.2 BDD and LWE

Definition 1 (BDD). *Let $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ be a full-rank basis of $\varLambda$, and χ_e be a distribution supported on $\mathbb{R}^n$ with norm concentrated around $\alpha\mathrm{GH}(n)\det(\varLambda)^{1/n}$ for some $\alpha \in (0,1)$. The search* Bounded Distance Decoding problem *(BDD) is,* *given on input the basis* $\boldsymbol{B}$ *and a target* $\boldsymbol{t} \in \mathbb{R}^n$ *generated by sampling* $\boldsymbol{e} \leftarrow \chi_e$ *and sampling* $\boldsymbol{c} \in \mathbb{Z}^n$ *from some distribution, and computing* $\boldsymbol{t} = \boldsymbol{B}\boldsymbol{c} + \boldsymbol{e}$*, to output* $(\boldsymbol{c}, \boldsymbol{e})$*.*

For sampled $\boldsymbol{e}$ of norm $\|\boldsymbol{e}\| \leq \frac{1}{2}\lambda_1(\Lambda)$, the only correct output is the pair $(\boldsymbol{v}, \boldsymbol{e})$. For $\alpha \in [\frac{1}{2}, 1)$, there is still one correct solution for a single $\boldsymbol{e}$ with a certain probability.

If $\boldsymbol{e}$ is sampled such that $\langle \mathbf{b}_i^*, \boldsymbol{e}\rangle \leq \frac{1}{2}\left\|\mathbf{b}_i^*\right\|^2$ holds for all $0 \leq i < n$, then NP solves that BDD instance. Considering a QR decomposition of basis $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$, this result is proven by Howgrave-Graham for the very similar BDD instance $(\boldsymbol{R}, \boldsymbol{Q}^\mathsf{T}\boldsymbol{t})$ [How07, Lemma 1]. This condition will be satisfied when α is sufficiently small and sufficient lattice basis reduction, i.e., LLL [LLL82] or BKZ [SE94,CN11], is performed on the basis $\boldsymbol{B}$.

The hardness of the BDD instance increases as α increases. A simple BDD instance can be probabilistically reduced to a harder instance, using NP. This reduction is shown abstractly in Algorithm 1.

Algorithm 1 BDDReduce$(\boldsymbol{B}, \boldsymbol{t}, n')$

```
Require: $(\boldsymbol{B}, \boldsymbol{t})$ is a BDD instance, and $n' < n$.
1: $\ell \leftarrow n - n'$
2: $(\boldsymbol{c}_2, \boldsymbol{e}') \leftarrow$ SearchBDD$(\boldsymbol{B}_{[\ell,n)}, \pi_\ell^\perp(\boldsymbol{t}))$
3: $(\boldsymbol{c}_1, \boldsymbol{e}) \leftarrow$ BabaiNP$\big(\boldsymbol{B}_{[0,\ell)}, \boldsymbol{t} - \boldsymbol{B}_{[0:d,\ell:n]}\boldsymbol{c}_2\big)$
4: $\boldsymbol{c} \leftarrow \begin{pmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{pmatrix}$
5: return $(\boldsymbol{c}, \boldsymbol{e})$
```

While the output $(\boldsymbol{c}, \boldsymbol{e})$ of Algorithm 1 always satisfies

$$\boldsymbol{t} = \boldsymbol{B}_{[0,\ell)}\boldsymbol{c}_1 + \boldsymbol{B}_{[0:d,\ell:n]}\boldsymbol{c}_2 + \boldsymbol{e} = \boldsymbol{B}\boldsymbol{c} + \boldsymbol{e},$$

it may not always successfully solve the search BDD problem.

Lemma 1. *Let* $(\boldsymbol{B}, \boldsymbol{t})$ *be a BDD instance generated from* $\boldsymbol{t} = \boldsymbol{v}^\circ + \boldsymbol{e}^\circ$*, with* $\boldsymbol{v}^\circ \in \Lambda$ *and* $\|\boldsymbol{e}^\circ\| \leq \frac{1}{2}\lambda_1(\Lambda)$*.*

If $|\langle \mathbf{b}_i^*, \boldsymbol{e}^\circ\rangle| \leq \frac{1}{2}\left\|\mathbf{b}_i^*\right\|^2$ *holds for all* $0 \leq i < n - n'$ *and if during execution of Algorithm 1, the subroutine* SearchBDD *returns* $(\boldsymbol{c}_2, \boldsymbol{e}')$ *such that* $\boldsymbol{c}_2$ *equals the last* n' *coefficients of* $\boldsymbol{B}^{-1}\boldsymbol{v}^\circ$*, then Algorithm 1 successfully solves BDD.*

Proof. The condition on SearchBDD implies that the remaining BDD problem on $(\boldsymbol{B}_{[0,\ell)}, \boldsymbol{t} - \boldsymbol{B}(0, \boldsymbol{c}_2))$ can be solved by Babai's NP using [How07, Lemma 1].

The Learning with Errors [Reg09] problem can be phrased in terms of BDD.

Definition 2 (LWE). *Let* $n, m, q \in \mathbb{Z}_{\geq 1}$*, and let* χ_s*,* χ_e *be distributions on* $(\mathbb{Z}/q\mathbb{Z})^n$ *and* $(\mathbb{Z}/q\mathbb{Z})^m$ *respectively. The* Learning with Errors problem *(LWE) is, given on input a pair* $(\boldsymbol{A}, \boldsymbol{b})$*, where the matrix* $\boldsymbol{A}$ *is sampled uniformly from* $(\mathbb{Z}/q\mathbb{Z})^{m\times n}$*, and* $\boldsymbol{b}$ *is obtained by sampling* $(\boldsymbol{s}, \boldsymbol{e}) \leftarrow (\chi_s, \chi_e)$ *and computing* $\boldsymbol{b} \equiv \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \pmod q$*, to output* $\boldsymbol{s}$*.*

We can interpret an LWE instance $(\boldsymbol{A}, \boldsymbol{b})$ as a BDD instance, by observing $\boldsymbol{b} - \boldsymbol{e} \in \mathcal{L}([q\,\mathrm{id}_m \,|\boldsymbol{A}])$. However, a better embedding exists taking advantage of the small secret. Namely, for any $\xi \in \mathbb{R}$, we can also interpret $(\boldsymbol{A}, \boldsymbol{b})$ as the BDD instance $(\boldsymbol{B}, \boldsymbol{t})$, where

$$\boldsymbol{B} = \begin{pmatrix} q\,\mathrm{id}_m & \boldsymbol{A} \\ \mathbf{0}^{n\times m} & \xi\,\mathrm{id}_n \end{pmatrix}, \qquad \boldsymbol{t} = \begin{pmatrix} \boldsymbol{b} \\ \mathbf{0}^n \end{pmatrix}. \tag{1}$$

Note, indeed we have $\boldsymbol{t} = \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{s} \end{pmatrix} + \begin{pmatrix} \boldsymbol{e} \\ -\xi\boldsymbol{s} \end{pmatrix}$ for a certain $\boldsymbol{u} \in \mathbb{Z}^m$. A good choice for ξ is one satisfying $\xi \left\|\boldsymbol{s}\right\| / \sqrt{n} \approx \left\|\boldsymbol{e}\right\| / \sqrt{m}$, which creates the easiest BDD distribution by finding the right balance between the determinant of Λ and the norm of $(\boldsymbol{e}, -\xi\boldsymbol{s})$ [BG14].

2.3 Sparse LWE Instances

The *Centred Binomial Distribution* $\mathcal{B}_\eta$ ($\eta \in \mathbb{Z}_{\geq 1}$) is the distribution obtained by sampling 2η uniform bits $b_1, \ldots, b_{2\eta} \in \{0, 1\}$, and outputting $(\sum_{i=1}^{2\eta} b_i) - \eta \in \{-\eta, \ldots \eta\}$. Moreover, $\mathcal{B}_\eta^{\mathrm{nz}}$ is the distribution that keeps sampling $x \leftarrow \mathcal{B}_\eta$ until $x \neq 0$, and then outputs that x.

The *Discrete Gaussian distribution* with parameter σ and support $\mathbb{Z}$ is denoted by $\mathcal{D}_\sigma$, and assigns to $x \in \mathbb{Z}$ a probability proportional to $\exp(-x^2/2\sigma^2)$ such that all its probabilities sum to 1.

Wenger et al. [WSM+25] are interested in the LWE problem using two specific types of distributions for (χ_s, χ_e), each having sparse secrets of a prescribed Hamming weight h, i.e., $\boldsymbol{s}$ has h nonzero entries.

Binomial with parameters $\eta, h \in \mathbb{Z}_{\geq 1}$:
- Error distribution χ_e consists of m i.i.d. samples from $\mathcal{B}_\eta$,
- Secret distribution χ_s is obtained by first sampling $I \subseteq \{1, 2, \ldots, n\}$ uniformly from all subsets of size h, and then returning $\boldsymbol{s} = (s_1, \ldots, s_n)$, where $s_i = 0$ if $i \notin I$ and $s_i \leftarrow \mathcal{B}_\eta^{\mathrm{nz}}$ otherwise.

Ternary with parameters $\sigma \in \mathbb{R}_{>0}$ and $h \in \mathbb{Z}_{\geq 1}$:
- Error distribution χ_e consists of m i.i.d. samples from $\mathcal{D}_\sigma$,
- Secret distribution χ_s is obtained by first sampling $I \subseteq \{1, 2, \ldots, n\}$ uniformly from all subsets of size h, and then returning $\boldsymbol{s} = (s_1, \ldots, s_n)$, where $s_i = 0$ if $i \notin I$ and $s_i \leftarrow \mathrm{U}(\{-1, 1\})$ otherwise.

We will refer to these types in short by “Bin” and “Ter”. By default, we use $\eta = 2$ in the Binomial case, and $\sigma = 3.19$ in the Ternary case. The prime q is chosen accoding to [WSM+25, Tables 5,6].

Module-LWE. Moreover, we will use Ring-LWE and Module-LWE instances. A Module-LWE instance of rank κ and degree D works with the number ring $\mathcal{R} = \mathbb{Z}[X]/(X^D + 1)$ for D a power of two, and the finite ring $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Following [WSM+25, Sec. 2.1], a *Module-LWE instance* is a pair $(\boldsymbol{A}, \boldsymbol{b}) \in \mathcal{R}_q^\kappa \times \mathcal{R}_q$

where $\boldsymbol{A} \leftarrow \mathrm{U}\big(\mathcal{R}_q^{\kappa}\big)$, and $\boldsymbol{b} \equiv \boldsymbol{A}^{\mathsf{T}}\boldsymbol{s} + \boldsymbol{e} \pmod q$ with secret vector $\boldsymbol{s} \in \mathcal{R}^{\kappa}$ and error vector $\boldsymbol{e} \in \mathcal{R}$. A *Ring-LWE instance* is a Module-LWE instance of rank $\kappa = 1$. A Module-LWE instance and Ring-LWE instance can both be considered as just one "structured" sample.

We consider the *coefficient embedding*, given by

$$\begin{aligned} \mathrm{vec}\colon \mathcal{R}/q\mathcal{R} &\to (\mathbb{Z}/q\mathbb{Z})^{D}, \\ c_0 + c_1 X \ldots + c_{D-1}X^{D-1} &\mapsto \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{D-1} \end{pmatrix}, \end{aligned}$$

and extended to modules $\mathrm{vec}\colon (\mathcal{R}/q\mathcal{R})^{\kappa} \to (\mathbb{Z}/q\mathbb{Z})^{\kappa \cdot D}$. Together with the rotation map

$$\begin{aligned} \mathrm{rot}\colon \mathcal{R}/q\mathcal{R} &\to (\mathbb{Z}/q\mathbb{Z})^{D \times D}, \\ r &\mapsto \left[\mathrm{vec}(r), \mathrm{vec}(rX), \ldots, \mathrm{vec}(rX^{D-1})\right], \end{aligned}$$

we can convert a Module-LWE instance to an unstructured LWE instance:

$$(\mathrm{rot}(\boldsymbol{A}), \mathrm{vec}(\boldsymbol{b}))\,,$$

is an LWE instance with secret dimension $n = \kappa \cdot D$, $m = D$ LWE samples, secret vector $\mathrm{vec}(\boldsymbol{s})$ and error $\mathrm{vec}(\boldsymbol{e})$.

2.4 Guess + Verify

The Guess + Verify attack generalises the Drop + Solve attack [KKN+25]. In this section, we assume familiarity with Drop + Solve. Both attacks are already analysed [ACW19]. The function `LWE.primal_hybrid` in the `lattice-estimator` tool [APS15] estimates the cost of Guess + Verify, when it is called with parameters `mitm=False` and `babai=True`.

The "Drop + Solve" attack has a number of coordinates $I \subseteq \{1, 2, \ldots, n\}$ from the secret that it keeps, i.e., $\boldsymbol{s}_I = (\boldsymbol{s}_i)_{i \in I}$, while k coordinates $J = \{1, 2 \ldots, n\} \setminus I$ are dropped from the secret $\boldsymbol{s}$. Instead of guessing that only zero coefficients are dropped, i.e., $\boldsymbol{s}_J = \boldsymbol{0}$, the Guess + Verify attack guesses that $\boldsymbol{s}_J$ has precisely $h' \in \{0, 1, \ldots, h\}$ nonzero values. The Drop + Solve attack is the particular instance of Guess + Verify having $h' = 0$. The Guess + Verify attack generally requires fewer iterations before finding the correct guess, although each iteration takes (slightly) more time. Namely, for a given set I, we get a batched BDD instance in which at most one target is from the BDD distribution. Algorithm 2 contains pseudocode for Guess + Verify.

To determine if a guess $\tilde{\boldsymbol{s}}_J$ is correct, first we compute $\boldsymbol{b}' = \boldsymbol{b} - \boldsymbol{A}_J\tilde{\boldsymbol{s}}_J$, and then we determine if $(\boldsymbol{A}_I, \boldsymbol{b}')$ is a LWE instance by converting it to a BDD instance using Equation (1). This can be done using a distinguishing criterion,

Algorithm 2 Guess + Verify$(\boldsymbol{A}, \boldsymbol{b}, h', k, \beta)$

```
Require:
    unstructured LWE instance $(\boldsymbol{A}, \boldsymbol{b})$ having secret of Hamming weight $h$,
    $h' \in \{0, 1, \ldots, h\}$,
    $k < n$ the number of dropped coordinates, and
    $\beta$ the maximum block size
 1: Set $N_{\text{iters}}$ according to the desired confidence level, see Section 2.4.
 2: for $i = 1, 2, \ldots, N_{\text{iters}}$ do
 3:     $I \leftarrow \mathrm{U}(\{S \subseteq \{1, 2, \ldots, n\} : |S| = n - k\})$
 4:     $J := \{1, 2, \ldots, n\} \setminus I$
 5:     $(\boldsymbol{A}_I, \boldsymbol{A}_J) := (\boldsymbol{A}_{[0:m,I]}, \boldsymbol{A}_{[0:m,J]})$
 6:     $\boldsymbol{B}_I := \begin{pmatrix} q\,\mathrm{id}_m & \boldsymbol{A}_I \\ \boldsymbol{0}^{(n-k)\times m} & \xi\,\mathrm{id}_{(n-k)} \end{pmatrix}$
 7:     $\boldsymbol{B}'_I \leftarrow \mathrm{ProgressiveBKZ}(\boldsymbol{B}_I, \beta)$
 8:     for all subsets $S \subseteq J$ of size $h'$ do
 9:         for all $\tilde{\boldsymbol{s}}_J \in \mathrm{Supp}(\chi_s)^k$ such that $\mathrm{Supp}(\tilde{\boldsymbol{s}}_J) = S$ do
10:             $\boldsymbol{t} := \begin{pmatrix} \boldsymbol{b} - \boldsymbol{A}_J \tilde{\boldsymbol{s}}_J \\ \boldsymbol{0}^{(n-k)} \end{pmatrix}$
11:             if $(\boldsymbol{B}'_I, \boldsymbol{t})$ is a BDD instance then
12:                 $(\boldsymbol{c}, \boldsymbol{e}) \leftarrow \mathrm{BabaiNP}(\boldsymbol{B}'_I, \boldsymbol{t})$
13:                 $\tilde{\boldsymbol{s}}_I := -\boldsymbol{e}_{[m:m+n-k]}/\xi$
14:                 return $\tilde{\boldsymbol{s}} = (\tilde{\boldsymbol{s}}_I, \tilde{\boldsymbol{s}}_J)$
15: return $\perp$
```

for example by checking if the error vector obtained after applying Babai's NP to the projected sublattice has small norm, see Section 3.2 for more details. The probability of a correct guess is

$$p = \frac{\binom{n-h}{k} \cdot \binom{h}{h'}}{\binom{n}{k}} \cdot p_{\text{babai}}$$

where p_{babai} is the probability that NP successfully solves the reduced BDD instance [ACW19]. The total number of iterations needed to find the correct guess with e.g. probability 99% is $N_{\text{iters}} = \lfloor \log(1 - 0.99)/\log(1 - p) \rceil$.

2.5 Modulus Switching using Centered Binomial Secrets

In this section, we generalize [KKN$^+$25, Heuristic 5.1] to secret vectors $\boldsymbol{s} \leftarrow \chi_s$ according to Binomial(η, h). Although modulus switching is not currently used for the experiments in Section 4, we include this for completeness, as it may be beneficial when attacking the ternary LWE instance with $\log_2(q) \approx 50$.

Definition 3 (Generalized Irwin–Hall Distributions). *Given h positive integers $\lambda_1, \ldots, \lambda_h \in \mathbb{Z}_{\geq 1}$, the* generalized centered Irwin–Hall distribution*, denoted by* $\mathrm{GCIH}(\lambda_1, \ldots, \lambda_h)$*, is the distribution of*

$$\sum_{i=1}^{h} \mathrm{U}\left(-\frac{\lambda_i}{2}, \frac{\lambda_i}{2}\right).$$

The probability density function is denoted by $f_{\mathrm{GCIH}(\lambda_1,\ldots,\lambda_h)}$.

The generalized discrete centered Irwin–Hall distribution*, which we denote by* $\mathrm{GCDIH}(\lambda_1,\ldots,\lambda_h)$*, has support* $S = [-\frac{s}{2}, \frac{s}{2}] \cap \mathbb{Z}$ *where* $s = \sum_{i=1}^{h} \lambda_i$*, and probability mass function* $f_{\mathrm{GCIH}}(x) / \sum_{y \in S} f_{\mathrm{GCIH}}(y)$ *at* $x \in S$.

Reusing the function $\varepsilon(x) = x - \lfloor x \rceil$ from [KKN+25, Appendix D], we heuristically assume the random variables $\varepsilon\left(\frac{p}{q}\boldsymbol{A}_{ij}\right)$ are i.i.d. with distribution $\mathrm{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$, similarly to [KKN+25, Appendix D].

Fix a given secret $\boldsymbol{s}$. Then,

$$\left\langle \varepsilon\left(\tfrac{p}{q}\boldsymbol{A}_i\right), \boldsymbol{s} \right\rangle \sim \sum_{j \in \mathrm{Supp}(\boldsymbol{s})} \boldsymbol{s}_j \, \mathrm{U}\left(-\frac{1}{2}, \frac{1}{2}\right) \sim \sum_{j \in \mathrm{Supp}(\boldsymbol{s})} \mathrm{U}\left(-\frac{|\boldsymbol{s}_j|}{2}, \frac{|\boldsymbol{s}_j|}{2}\right),$$

where we used $\lambda \, \mathrm{U}(a,b) = \mathrm{U}(\lambda a, \lambda b)$ ($\lambda > 0$).

Let us write $\lambda_1, \ldots, \lambda_h$ for the h values $|\boldsymbol{s}_j|$ with $j \in \mathrm{Supp}(\boldsymbol{s})$. Now, we heuristically have,

$$\begin{aligned} \tilde{\boldsymbol{e}}_i &\sim \frac{p}{q}\boldsymbol{e}_i + \sum_{j \in \mathrm{Supp}(\boldsymbol{s})} \mathrm{U}\left(-\frac{|\boldsymbol{s}_j|}{2}, \frac{|\boldsymbol{s}_j|}{2}\right) + \mathrm{U}\left(-\tfrac{1}{2}, \tfrac{1}{2}\right) \quad \bmod^{\pm} p \\ &\approx \mathrm{GCIH}(1, \lambda_1, \ldots, \lambda_h) \quad \bmod^{\pm} p \\ &= \mathrm{GCIH}(1, \lambda_1, \ldots, \lambda_h) \end{aligned}$$

where we use $p\sigma_e \ll q$ and $1 + \sum_j |\boldsymbol{s}_j| < p$ in the first and second equality respectively. Because $\tilde{\boldsymbol{e}}_i$ is integral by definition, we conjecture in practice

$$\tilde{\boldsymbol{e}}_i \;\sim\; \mathrm{GCDIH}(1, \lambda_1, \ldots, \lambda_h).$$

Moreover, $\mathbb{E}[\tilde{\boldsymbol{e}}_i] = 0$, and by independence

$$\mathrm{Var}(\tilde{\boldsymbol{e}}_i) \;\approx\; \frac{1}{12}\left(1 + \sum_{j \in \mathrm{Supp}(s)} |\boldsymbol{s}_j|^2\right) = \frac{1 + \|\boldsymbol{s}\|^2}{12}.$$

If $\boldsymbol{s}$ is sampled from χ_s according to $\mathrm{Binomial}(\eta, h)$, then

$$\mathbb{E}[\|\boldsymbol{s}\|^2] = h \cdot \mathbb{E}_{x \leftarrow \mathcal{B}_\eta^{\mathrm{nz}}}[x^2] = h \cdot \mathbb{E}_{x \leftarrow \mathcal{B}_\eta}[x^2 | x \neq 0] = h \frac{\eta/2}{1 - \binom{2\eta}{\eta}/4^\eta},$$

because a sample from $\mathcal{B}_\eta$ is zero with probability $\binom{2\eta}{\eta}/4^\eta$.

In the case $\eta = 2$, one has $\mathbb{E}_{x \leftarrow \mathcal{B}_\eta^{\mathrm{nz}}}[x^2] = \frac{8}{5}$, and

$$\mathrm{Var}(\tilde{\boldsymbol{e}}_i) \;\approx\; \frac{1}{12}\,(1 + 1.6h).$$

3 Lattice reduction on GPUs

To reduce the lattices arising in the primal attack, we offload the dominant dense linear-algebra primitives to GPUs by porting `BLASter` [DPS25]; we refer to the resulting system as `cuBLASter`. Our implementation relies on CuPy, whose NumPy-compatible API dispatches to cuBLAS and cuSOLVER. We use QR factorizations, general matrix-matrix multiplications (GEMM), and related routines, complemented by custom element-wise kernels where library coverage is insufficient. While we only explain the most notable changes, the full implementation is found on GitHub, see Section 1.1. The local LLL/BKZ reductions are done on the CPU. Section 3.1 explains how Seysen's reduction [Sey93, Proposition 5] is done on the GPU.

In addition, Section 3.2 details a GPU-version of Babai's Nearest Plane Algorithm [Bab86], using the batched variant [DPS25, Appendix A]. This is used to determine the correct guess $\tilde{\boldsymbol{s}}_J$ of weight h' in the primal hybrid attack.

3.1 Seysen's Reduction on the GPU

Seysen originally defined this reduction method [Sey93, Proposition 5] with a loop approach, but a Seysen-like reduction defined recursively was proposed in [KEF21, Section 3.4] and used in [DPS25]. This recursive method can be implemented iteratively [DPS25, Algorithm 1], see the file `src/size_reduction.py` in the implementation of [DPS25]. We describe here how we adapted it to the GPU, the main idea is to reduce many submatrices of the same size in batch. Given an upper-triangular matrix $\boldsymbol{R}' \in \mathbb{R}^{n\times n}$, we parse

$$\begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ \boldsymbol{0} & \boldsymbol{R}_{22} \end{bmatrix} \leftarrow \boldsymbol{R}',$$

where $\boldsymbol{R}_{11}$ has size $\frac{1}{2} \cdot 2^{\lceil \log_2(n) \rceil}$, instead of size $\lfloor n/2 \rfloor$, cf. [DPS25, Algorithm 1]. This makes the GPU code more efficient: because we mostly encounter submatrices of $\boldsymbol{R}'$ having size a power of two, we can easily batch operations that act on different parts of the matrix.

Suppose we want to perform Seysen reduction on $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ in-place, and want as output the transformation matrix $\boldsymbol{U} \in \mathbb{Z}^{n\times n}$ such that the final output is $\boldsymbol{RU}$. First, we initialize $\boldsymbol{U} \leftarrow \mathrm{id}_n$. For $k = 2, 4, 8, \ldots, 2^{\lceil \log_2(n) \rceil}$, we consider the following list of submatrices of $\boldsymbol{R}$:

$$\boldsymbol{R}_{[0:k,\,0:k]}, \quad \boldsymbol{R}_{[k:2k,\,k:2k]}, \quad \boldsymbol{R}_{[2k:3k,\,2k:3k]}, \quad \ldots, \quad \boldsymbol{R}_{[uk:n,\,uk:n]}.$$

Then, for a given submatrix $\boldsymbol{R}' = \boldsymbol{R}_{[i:j,i:j]}$ we execute Lines 3,6,7,8 from [DPS25, Algorithm 1] on $\boldsymbol{R}'$, while parsing similarly

$$\begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ \boldsymbol{0} & \boldsymbol{U}_{22} \end{bmatrix} \leftarrow \boldsymbol{U}_{[i:j,i:j]},$$

and the final result $\boldsymbol{U}_{12} \leftarrow \boldsymbol{U}_{11}\boldsymbol{U}'_{12}$ is then stored in that respective submatrix of $\boldsymbol{U}$.

Note that all submatrices have dimension $k \times k$, which we call the *interior submatrices*, except for the last submatrix of dimension $(n - uk) \times (n - uk)$ (where $u = \lfloor n/k \rfloor$), which we call the *terminal submatrix*. The reduction of the interior submatrices is handled in batch, while the terminal submatrix is handled separately. This terminal submatrix is not reduced when $uk + \frac{k}{2} \geq n$.

Hence, per level k we issue at most two partial reductions: one batch for all interior submatrices and possibly one for the terminal submatrix. The first level, $k = 2$, can be handled by a vectorized super-diagonal update. This yields at most $2\lceil \log_2 n \rceil$ kernel sequences overall. In practice, there are $2\lceil \log_2 n \rceil - 1$ kernel sequence calls for $k \geq 4$, plus the super-diagonal update.

To control launch overhead, when there are less than 4 interior submatrices or 8 interior submatrices if $\boldsymbol{U}_{12}$ contains less than 4096 entries, we fall back to an unbatched path with the same algebraic updates.

3.2 Babai's Nearest Plane on the GPU

To solve many approximate BDD instances in parallel for the primal hybrid attack, we implement a GPU version of the batched, projected variant of Babai's nearest-plane (NP) algorithm [Bab86]. Let $\Lambda = \mathcal{L}(\boldsymbol{B}) \subset \mathbb{R}^n$ be a full-rank lattice with basis $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ and let $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m] \in \mathbb{R}^{n \times m}$ be a matrix of target vectors. We compute the QR factorization $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q}$ is orthonormal and $\boldsymbol{R}$ is upper triangular.

Instead of applying Babai's algorithm to the full basis, we project to the last $\ell \leq n$ coordinates. We partition $\boldsymbol{R}$ and $\boldsymbol{Q}^\top$ as

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ \boldsymbol{0} & \boldsymbol{R}_{22} \end{bmatrix}, \quad \boldsymbol{Q}^\top = \begin{bmatrix} \boldsymbol{Q}_1^\top \\ \boldsymbol{Q}_2^\top \end{bmatrix}, \qquad \text{where} \quad \boldsymbol{R}_{22} \in \mathbb{R}^{\ell \times \ell}, \ \boldsymbol{Q}_2^\top \in \mathbb{R}^{\ell \times n}.$$

The targets are projected into the ℓ-dimensional subspace by computing

$$\boldsymbol{Y} \; = \; \boldsymbol{Q}_2^\top \boldsymbol{T} \; \in \; \mathbb{R}^{\ell \times m}.$$

Running the batched NP algorithm on the pair $(\boldsymbol{R}_{22}, \boldsymbol{Y})$ returns integer coefficient vectors $\boldsymbol{U}_{\text{bot}} \in \mathbb{Z}^{\ell \times m}$ and a matrix $\boldsymbol{E} = \boldsymbol{Y} - \boldsymbol{R}_{22}\boldsymbol{U}_{\text{bot}} \in \mathbb{R}^{\ell \times m}$. A candidate i is accepted if there exists a vector $\boldsymbol{e}_i$ in $\boldsymbol{E}$ whose squared norm $\|\boldsymbol{e}_i\|^2$ is below a certain threshold τ_{proj}^2. For each accepted candidate, we reconstruct the full solution vector by running Babai's algorithm on the full basis $\boldsymbol{B}$ on the CPU, using `fplll` [FPL23]. We use `fplll` as fallback to avoid any numerical issues based on the floating point precision of the GPU. We could also implement the full Babai's algorithm on the GPU, and if it doesn't have numerical errors, we can skip the `fplll` step, but we prefer to set this GPU fallback on the whole basis as a fallback of the `fplll` step to be sure we don't miss any solution by accident in the check of the numerical errors in $\boldsymbol{E}$.

Verifying a Guess on the GPU. Parallelism is achieved in Guess + Verify across the candidate guesses. First, we guess that, out of k coordinates of the secret $\boldsymbol{s}$, there are h' nonzero. Specifically, we sample a subset $J \subseteq \{1, \ldots, n\}$ of size

k uniformly at random. Then, to verify this guess, we enumerate all possible values $\boldsymbol{s}_J = (\boldsymbol{s}_i)_{i \in J}$ may have. Namely, we iterate over all $\tilde{\boldsymbol{s}}_J \in \mathrm{Supp}(\chi_s)^J$ of Hamming weight h'. The corresponding target vector is $\boldsymbol{b} - \boldsymbol{A}_J \tilde{\boldsymbol{s}}_J$, where $(\boldsymbol{A}, \boldsymbol{b})$ is the original LWE instance, and $\boldsymbol{A}_J$ consists of the columns of $\boldsymbol{A}$ having an index in J. During one iteration, the total number of considered candidates $\tilde{\boldsymbol{s}}_j$ equals

$$G = \binom{k}{h'} \cdot (|\mathrm{Supp}(\chi_s)| - 1)^{h'},$$

where we remark that exactly h' values must come from the set $\mathrm{Supp}(\chi_s) \setminus \{0\}$. For example, $G = \binom{k}{h'} 2^{h'}$ when the instance is ternary, see Section 2.3.

To use this full potential of parallelism while managing limited GPU VRAM, we process candidates in batches with a given size. An *index batch* contains at most $B_i =$ `GUESS_BATCH` choices of coordinate sets, and a *value batch* contains at most $B_v =$ `VALUE_BATCH` value assignments; in practice, B_v was always set higher than $\mathrm{Supp}(\chi_s)^{h'}$, so it is capped at that value. The total number of candidates per iteration is the batch size $B = B_i \cdot B_v$. All GPU kernels operate on matrices with B columns.

Let $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$ and $\boldsymbol{b} \in \mathbb{Z}_q^m$. For an index batch, we gather the h' columns of $\boldsymbol{A}$ corresponding to each of the B_i index sets into a 3-dimensional tensor $\boldsymbol{A}_{\text{guess}} \in \mathbb{R}^{m \times B_i \times h'}$. Concretely, for the i-th index set $J_i \subseteq \{1, \ldots, n\}$ with $|J_i| = h'$, the slice $\boldsymbol{A}_{\text{guess}}[:, i, :]$ contains the submatrix $\boldsymbol{A}_{[:, J_i]} \in \mathbb{R}^{m \times h'}$. For a value batch, we stack the B_v value vectors into $\boldsymbol{V} \in \mathbb{R}^{h' \times B_v}$. To compute all $B = B_i \cdot B_v$ products $\boldsymbol{A}_{[:, J_i]} \boldsymbol{v}_j$ in parallel (for $i \in \{1, \ldots, B_i\}$ and $j \in \{1, \ldots, B_v\}$), we reshape the tensor $\boldsymbol{A}_{\text{guess}}$ into a 2D matrix by stacking the B_i slices vertically and perform a single batched matrix multiplication:

$$\underbrace{(\boldsymbol{A}_{\text{guess}})_{(mB_i) \times h'}}_{\text{row-major view}} \cdot \underbrace{\boldsymbol{V}_{h' \times B_v}}_{\text{values}} = \underbrace{\boldsymbol{G}_{(mB_i) \times B_v}}_{\text{products}}.$$

The resulting matrix $\boldsymbol{G}$ is reshaped to size $m \times B$. After forming the target vectors $\boldsymbol{b}\boldsymbol{1}_B^\top - \boldsymbol{G}$ (where $\boldsymbol{1}_B$ is the all-ones vector of length B) and reducing modulo q, we project them all with a single matrix multiplication:

$$\boldsymbol{Y} = \boldsymbol{Q}_2^\top (\boldsymbol{b}\boldsymbol{1}_B^\top - \boldsymbol{G}) \in \mathbb{R}^{\ell \times B}.$$

Selecting the Projected Dimension ℓ. The choice of projection dimension ℓ balances computational cost against the true-positive and false-positive rates.

Suppose the algorithm enumerates G incorrect guesses. We will heuristically model a target belonging to such incorrect guess by a uniform sample from $\mathbb{R}^\ell / \mathcal{L}(\boldsymbol{R}_{22})$. The expected number of false positives with norm at most τ is

$$G \cdot \frac{\mathrm{Vol}_\ell(\tau \mathcal{B}^\ell)}{\det(\Lambda_{\text{proj}})} = G \cdot \frac{(\tau / \mathrm{GH}(\ell))^\ell}{\det |\boldsymbol{R}_{22}|}. \tag{2}$$

Our goal is to bound the expected number of false positives by p_{FP}, which lower bounds the probability of having no false positives by $1 - p_{\text{FP}}$ using

Markov's inequality. Using Equation (2), we see that the expected number of false positives is p_{FP}, if the norm bound τ satisfies

$$\tau \leq \mathrm{GH}(\ell) \cdot \left(\frac{p_{\mathrm{FP}} \cdot \det |\boldsymbol{R}_{22}|}{G}\right)^{1/\ell}. \tag{3}$$

On the other hand, we want the norm bound τ to be large enough to detect the correct guess with a probability of at least p_{TP}. For the true positive, the projected noise vector $\boldsymbol{z}$ follows $\mathcal{N}(0, \sigma^2 \boldsymbol{I}_\ell)$, so $\|\boldsymbol{z}\|^2/\sigma^2$ follows a χ^2_ℓ distribution. To capture the correct solution with probability p_{TP}, our norm bound τ must satisfy

$$\tau \geq \sigma \sqrt{q_{\chi^2_\ell}(p_{\mathrm{TP}})}. \tag{4}$$

Combining Equations (3) and (4), we select the smallest dimension ℓ that satisfies

$$\sigma \sqrt{q_{\chi^2_\ell}(p_{\mathrm{TP}})} < \mathrm{GH}(\ell) \cdot \left(\frac{p_{\mathrm{FP}} \cdot \det |\boldsymbol{R}_{22}|}{G}\right)^{1/\ell}.$$

Then, we pick any τ in between both sides, which ensures we guess correctly with probability at least $p_{\mathrm{TP}} - p_{\mathrm{FP}}$ by a union bound. Moreover, this reduces the cost of the batched NP algorithm.

Specifically, we pick $p_{\mathrm{TP}} = 0.99$ and $p_{\mathrm{FP}} = 0.01$.

4 Results

In this Section we benchmark cuBLASter and the Guess + Verify attack.

Section 4.1 covers the used hardware for our experiments. In Section 4.2, we benchmark cuBLASter on the same machine and instances BLASter used [DPS25], and compare cuBLASter to BLASter [DPS25], `fplll` [FPL23] and `flatter` [RH23]. In Section 4.3 we explain how the LWE instances are generated. Finally, in Section 4.4 we benchmark the Guess + Verify attack on the instances used to benchmark the Cool & Cruel [NMW+24] attack [WSM+25].

4.1 Hardware specifications

Table 1. Hardware used for our experiments. Some experiments used fewer CPUs and GPUs than available.

Name	CPU (quantity × logical cores)	GPU (quantity)
Y	AMD EPYC-Milan @2.745GHz (1 × 96)	NVIDIA H100 (1)
H	Intel Xeon Gold 6248 @2.5GHz (2 × 40)	NVIDIA RTX 2080 Ti (4)
Z	Intel Xeon Gold 5222 @3.8GHz (2 × 8)	NVIDIA RTX 2080 Ti (8)

Table 1 lists all the machines we used for benchmarking. We have benchmarked cuBLASter on the same machine 'H' as BLASter [DPS25]. While the server contains 4 GPUs, only 2 were used. The Guess + Verify attack ran on all three machines.

In comparison, Wenger *et al.* [WSM$^+$25] conducted benchmarks on a whole computing cluster having "2.1GHz Intel Xeon Gold CPUs and NVIDIA V100 GPUs". The iterations in Guess + Verify can run in parallel, so running the attack with more GPUs would lower wall times. Hence, for a fair comparison between different hardware configurations, we focus on the total number of core-hours and GPU-hours instead of wall clock time.

4.2 Lattice Reduction Experiments

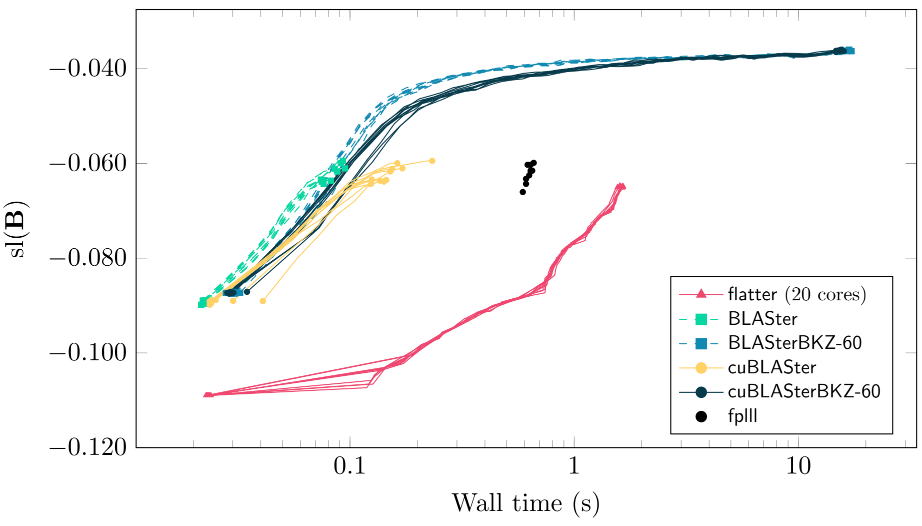

Fig. 1. Slope as function of used wall time for 10 random 631-ary 128-dimensional lattices. Lower slope indicates better basis quality. Comparison of `flatter`, `fplll`, BLASter, and cuBLASter (this work) on machine H with 2 CPUs (40 cores each) and 1 GPU.

First, Figures 1, 2, 3 and 4 show the performance of cuBLASter on the benchmarks used in BLASter [DPS25], using a single GPU on the server 'H'. In both cases $\lceil n/64 \rceil$ cores were used to reduce a lattice of dimension n. The *slope of the reduced basis*, $\mathrm{sl}\,\boldsymbol{B}$, is plotted as a function of the wall time. For a definition of the slope associated to a basis, see [DPS25].

For completeness, we report the differences in Table 2 In dimension 128 and 256, cuBLASter compares to BLASter in terms of wall time and reduction quality. However, for large dimensions, cuBLASter is at least *twice (resp. four times) as*

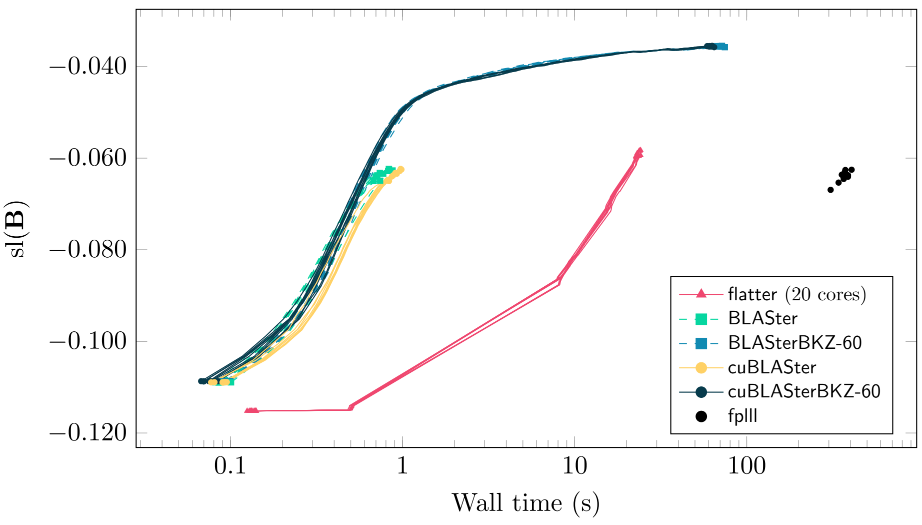

Fig. 2. Slope as function of used wall time for 10 random 829561-ary 256-dimensional lattices. Lower slope indicates better basis quality. Comparison of `flatter`, `fplll`, BLASter, and cuBLASter (this work) on machine H.

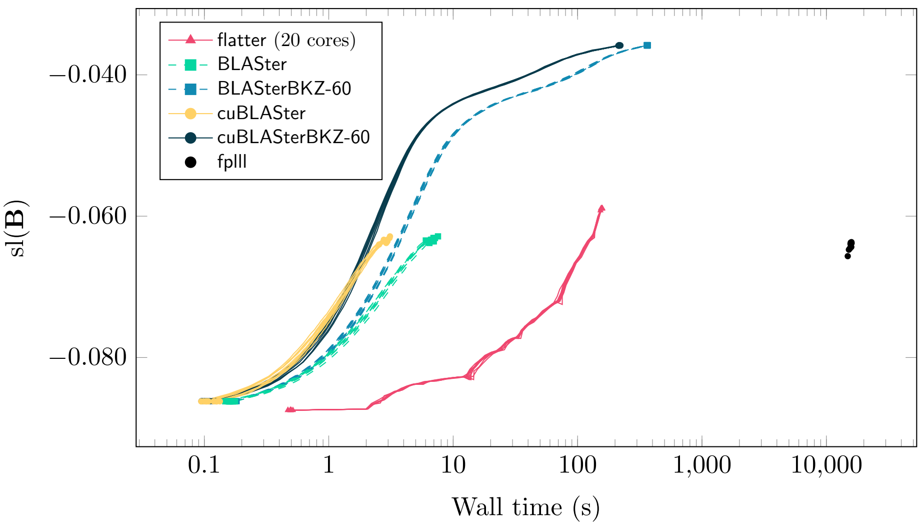

Fig. 3. Slope as function of used wall time for 10 random 968665207-ary 512-dimensional lattices. Lower slope indicates better basis quality. Comparison of `flatter`, `fplll`, BLASter, and cuBLASter (this work) on machine H.

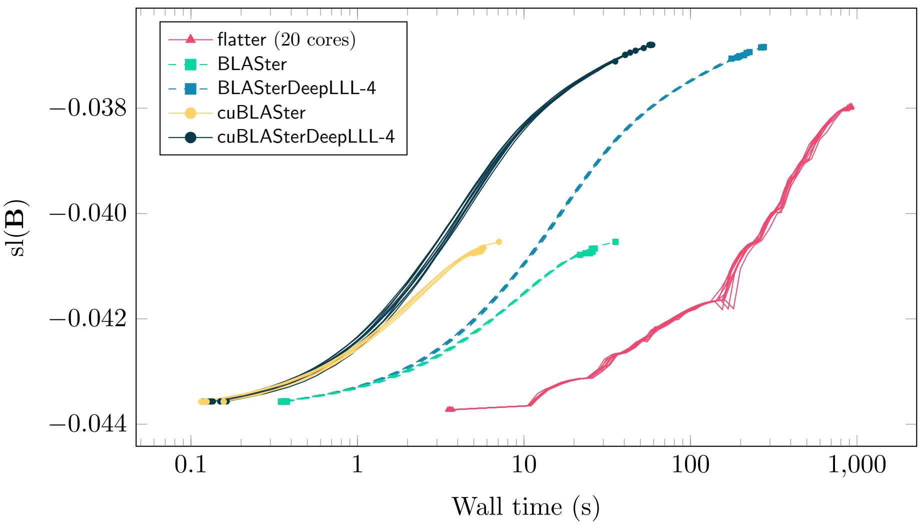

Fig. 4. Slope as function of used wall time for 10 random 968665207-ary 1024-dimensional lattices. Lower slope indicates better basis quality. We gave `flatter` the argument $\alpha = 0.043$ to make sure the basis is reduced until $\text{sl}(\boldsymbol{B}) > -0.043$ holds. Comparison of `flatter`, BLASter, and cuBLASter (this work) on machine H.

Table 2. Comparison between runtime of `flatter`, BLASter, and cuBLASter (this work) on machine H. "Algorithm" specifies which lattice reduction variant is used in BLASter and cuBLASter. No extra parameters were given to `flatter` [RH23], except $\alpha = 0.043$ in dimension 1024. The time of `flatter` is given in cases when reduction quality compares to that of BLASter. The achieved reduction quality (slope) can be found in Figures 1, 2, 3 and 4. Bold entries indicate the fastest method for each configuration.

		Wall time		
Algorithm	Dimension	`flatter`	BLASter	cuBLASter
LLL	128	1628 ms	**84 ms**	154 ms
LLL	256	23 s	**770 ms**	911 ms
LLL	512	156 s	6.8 s	**2.9 s**
LLL	1024		26.0 s	**5.6 s**
BKZ-60	128		16 s	**15 s**
BKZ-60	256		70 s	**61 s**
BKZ-60	512		363 s	**218 s**
DeepLLL-4	1024	904 s	223 s	**49 s**

fast as BLASter in dimension 512 (resp. 1024). Progressive BKZ-60 in dimension 512, becomes 40% faster when using cuBLASter instead of BLASter.

4.3 Instance Generation

The Guess + Verify attack runs on LWE parameters specified by [WSM$^+$25]. Namely, we consider binomial and ternary instances explained in Section 2.3. The LWE instances are generated using a pseudorandom number generator (PRNG) that is fed by a fixed seed, and are structured, i.e., we generate a Module-LWE instance.

Wenger *et al.* [WSM$^+$25] considered an attack successful if it succeeded at least once on 10 instances, and if so, reports the lowest wall time of each successful run. Because Guess + Verify and Cool & Cruel are probabilistic attacks, their wall times may vary a lot depending on the seed used to generate the LWE instance, e.g., the attack can be extremely "lucky" by finding the secret in the first iteration. Hence, we report for each parameter set, not the minimum but the average wall time of all successful runs.

4.4 Experimental Results

Our results in Table 3 suggest that Guess + Verify outperforms C+C for every tested parameter set.

Remark 1. The reported minimal computation time for Cool & Cruel in Table 3 is taken from [WSM$^+$25, Table 9,11,12]. The reported wall times from [WSM$^+$25] only have 1 or 2 significant digits in some cases, so we report the core-hours and GPU-hours while taking into account the uncertainty of the reported wall times.

For example, the first column [WSM$^+$25, Table 9] reports that Cool & Cruel ran for 0.1 hours on 256 GPUs. Since the precise wall time is anywhere between 0.05 and 0.15 hours, the number of GPU hours used is in the range $[12.8, 38.4]$. The reported mean is obtained by multiplying wall time with the number of GPUs. Therefore, the first row in Table 3 reports Cool & Cruel used at least 26 ± 13 GPU-hours on one successful run.

First, C+C does not achieve higher success probability than Guess + Verify in any of the tested parameter sets, and Guess + Verify always succeeds for a ternary instance where C+C always fails [WSM$^+$25, Table 12]. Moreover, Guess + Verify solved a binomial instance of higher Hamming weight than achievable by C+C in [WSM$^+$25, Table 9].

In addition, Guess + Verify's GPU-based recovery step is cheaper than C+C's. The total GPU utilization of C+C measured in GPU $\cdot$ wall hours is at least double that of Guess + Verify. C+C only uses GPUs for the recovery step, while Guess + Verify uses GPUs for lattice reduction and recovery, so the GPU utilization in Guess + Verify's recovery step is arguably lower than what is listed in Table 3. Moreover, only C+C's minimal GPU utilization is reported [WSM$^+$25], so Guess + Verify's average performance is likely even better when compared with C+C's average performance. Unfortunately, full runtime measurements for the experiments in [WSM$^+$25] were not published. Even if we exclude the cost of lattice reduction from C+C, Guess + Verify's total GPU utilization (including lattice reduction) is still lower than that of C+C on every tested parameter set.

A different topic is the preprocessing step, which involves lattice reduction. The Guess + Verify attack uses cuBLASter for lattice reduction. The cuBLASter library appears to be an order of magnitude faster than the combination of FPLLL, `flatter` [RH23] and `polish` [CLLT24] used by Cool & Cruel. This cuBLASter library only intensively uses the CPU for pruned enumeration inside BKZ. We report the theoretical maximum CPU utilization for completeness, although precise CPU utilization is much lower.

Table 3. Experimental comparison of Guess + Verify (this work) and Cool & Cruel [NMW+24,WSM+25] against sparse secret LWE instances. The Guess + Verify attack ran on servers from Table 1: Y used 92 CPU cores and 1 GPU, Z used 8 CPU cores and 8 GPUs, and H used 40 CPU cores and 2 GPUs. LWE types "Bin" (Binomial) and "Ter" (Ternary) refer to Section 2.3. Attack parameters are explained in Section 2.4. "succ." denotes the number of successful runs out of total attempts. For Guess + Verify, we report average timings over successful runs; for Cool & Cruel, we report minimum timings from [WSM+25] with uncertainty bounds.

Module-LWE params				**Guess + Verify**									**Cool & Cruel**		
				Attack parameters					avg.	avg.	avg.			min.	min.
Type	n	q	h	β	m	k	h'	succ.	wall time (s)	core-hours	GPU-hours	server	succ.	core-hours	GPU-hours
Bin	$2 \cdot 256$	$\approx 2^{12}$	11	46	72	393	3	5/6	20 448	≤ 522.5	5.7	Y	2/10	4508 ± 81	26 ± 13
Bin	$2 \cdot 256$	$\approx 2^{12}$	12	46	72	395	3	4/5	108 516	≤ 2773.2	30.1	Y		*could not run*	
Bin	$2 \cdot 256$	$\approx 2^{28}$	20	52	192	234	≤ 3	4/5	10 158	≤ 22.6	22.6	Z	3/10	1661 ± 76	51 ± 13
Bin	$2 \cdot 256$	$\approx 2^{28}$	21	52	192	235	≤ 3	5/5	12 547	≤ 27.9	27.9	Z	3/10	1661 ± 76	154 ± 13
Bin	$2 \cdot 256$	$\approx 2^{28}$	25	53	192	235	≤ 3	5/5	73 790	≤ 164.0	164.0	Z	1/10	1661 ± 76	10752 ± 128
Ter	$1 \cdot 1024$	$\approx 2^{26}$	11	49	197	768	3	5/5	32 584	≤ 832.7	9.1	Y	1/10	1856 ± 29	102 ± 52
Ter	$1 \cdot 1024$	$\approx 2^{29}$	9	53	274	800	3	5/5	16 841	≤ 187.1	9.4	H	10/10	1833 ± 3	31 ± 6
Ter	$1 \cdot 1024$	$\approx 2^{29}$	10	53	274	801	3	5/5	76 073	≤ 845.3	42.3	H	0/10	1833 ± 3	*no success*

References

ACC$^+$18. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. URL: `https://homomorphicencryption.org/standard/`.

ACW19. Martin R. Albrecht, Benjamin R. Curtis, and Thomas Wunderer. Exploring trade-offs in batch bounded distance decoding. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 467–491. Springer, Cham, August 2019. `doi:10.1007/978-3-030-38471-5_19`.

ADH$^+$19. Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Cham, May 2019. `doi:10.1007/978-3-030-17656-3_25`.

AGVW17. Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 297–322. Springer, Cham, December 2017. `doi:10.1007/978-3-319-70694-8_11`.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. `doi:10.1515/jmc-2015-0016`.

Bab86. László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. `doi:10.1007/BF02579403`.

BCC$^+$24. Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez, Yongsoo Song, and Donggeon Yhee. Security guidelines for implementing homomorphic encryption. *IACR Commun. Cryptol.*, 1(4):26, 2024. `doi:10.62056/ANXRA69P1`.

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. `doi:10.1137/1.9781611974331.ch2`.

BG14. Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 322–337. Springer, Cham, July 2014. `doi:10.1007/978-3-319-08344-5_21`.

BGPW16. Johannes A. Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 24–43. Springer, Cham, April 2016. `doi:10.1007/978-3-319-31517-1_2`.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. doi:10.1145/2090236.2090262.

BLLW22. Lei Bi, Xianhui Lu, Junjie Luo, and Kunpeng Wang. Hybrid dual and meet-LWE attack. In Khoa Nguyen, Guomin Yang, Fuchun Guo, and Willy Susilo, editors, *ACISP 22*, volume 13494 of *LNCS*, pages 168–188. Springer, Cham, November 2022. doi:10.1007/978-3-031-22301-3_9.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_50.

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78381-9_12.

CHK$^+$18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78381-9_14.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017. doi:10.1007/978-3-319-70694-8_15.

CLLT24. François Charton, Kristin Lauter, Cathy Li, and Mark Tygert. An efficient algorithm for integer lattice reduction. *SIAM Journal on Matrix Analysis and Applications*, 45(1):353–367, 2024. doi:10.1137/23M1557933.

CN11. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, December 2011. doi:10.1007/978-3-642-25385-0_1.

DPS23. Léo Ducas, Eamonn Postlethwaite, and Jana Sotáková. Salsa Verde versus the actual state of the art. In *Rump Session at CRYPTO'23*, August 2023. URL: https://crypto.iacr.org/2023/rump/crypto2023rump-paper13.pdf.

DPS25. Léo Ducas, Ludo N. Pulles, and Marc Stevens. Towards a modern LLL implementation. Cryptology ePrint Archive, Paper 2025/774, 2025. To appear at ASIACRYPT'25. URL: https://eprint.iacr.org/2025/774.

DSvW21. Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden. Advanced lattice sieving on GPUs, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 249–279. Springer, Cham, October 2021. doi:10.1007/978-3-030-77886-6_9.

Duc18. Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78381-9_5.

FPL23. The FPLLL development team. fplll, a lattice reduction library, Version: 5.5.0. Available at https://github.com/fplll/fplll, 2023. URL: https://github.com/fplll/fplll.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

GNR10. Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_13.

HGSW03. Nick Howgrave-Graham, Joseph H Silverman, and William Whyte. A meet-in-the-middle attack on an NTRU private key. Technical report, NTRU Cryptosystems, June 2003. URL: https://www.researchgate.net/publication/2906622_A_Meet-In-The-Middle_Attack_on_an_NTRU_Private_Key.

HKLS22. Minki Hhan, Jiseung Kim, Changmin Lee, and Yongha Son. A hybrid of lattice-reduction and meet-LWE via near-collision on babai's plane. Cryptology ePrint Archive, Paper 2022/1473, 2022. URL: https://eprint.iacr.org/2022/1473.

How07. Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, Berlin, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_9.

HPS98. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, June 1998.

HS21. Shai Halevi and Victor Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, January 2021. doi:10.1007/s00145-020-09368-7.

KEF21. Paul Kirchner, Thomas Espitau, and Pierre-Alain Fouque. Towards faster polynomial-time lattice reduction. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 760–790, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84245-1_26.

KKN$^+$25. Alexandr Karenin, Elena Kirshanova, Julian Nowakowski, Eamonn W. Postlethwaite, and Fernando Virdia. Cool + cruel = dual. Cryptology ePrint Archive, Paper 2025/1002, 2025. URL: https://eprint.iacr.org/2025/1002.

LDK$^+$22. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

LLL82. Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982. doi:10.1007/BF01457454.

LWAZ$^+$23. Cathy Li, Emily Wenger, Zeyuan Allen-Zhu, Francois Charton, and Kristin E. Lauter. SALSA VERDE: a machine learning attack on LWE with sparse small secrets. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 53343–53361. Curran Associates, Inc., 2023.

URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/a75db7d2ee1e4bee8fb819979b0a6cad-Paper-Conference.pdf.

May21. Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 701–731, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84245-1_24.

Ngu21. Phong Q. Nguyen. Boosting the hybrid attack on NTRU: torus LSH, permuted HNF and boxed sphere. NIST Third PQC Standardization Conference, June 9–11, 2021. URL: https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/nguyen-boosting-hybridboost-pqc2021.pdf.

NMW$^+$24. Niklas Nolte, Mohamed Malhou, Emily Wenger, Samuel Stevens, Cathy Yuanchen Li, François Charton, and Kristin E. Lauter. The cool and the cruel: Separating hard parts of LWE secrets. In Serge Vaudenay and Christophe Petit, editors, *AFRICACRYPT 24*, volume 14861 of *LNCS*, pages 428–453. Springer, Cham, July 2024. doi:10.1007/978-3-031-64381-1_19.

PFH$^+$22. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005. doi:10.1145/1568318.1568324.

RH23. Keegan Ryan and Nadia Heninger. Fast practical lattice reduction through iterated compression. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 3–36. Springer, Cham, August 2023. doi:10.1007/978-3-031-38548-3_1.

SAB$^+$22. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

SE94. Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994. Preliminary version in FCT 1991. doi:10.1007/BF01581144.

Sey93. Martin Seysen. Simultaneous reduction of a lattice basis and its reciprocal basis. *Combinatorica*, 13(3):363–376, 1993. doi:10.1007/BF01202355.

SWL$^+$25. Samuel Stevens, Emily Wenger, Cathy Yuanchen Li, Niklas Nolte, Eshika Saxena, Francois Charton, and Kristin E. Lauter. Salsa Fresca: Angular embeddings and pre-training for ML attacks on learning with errors. *Transactions on Machine Learning Research*, 2025. URL: https://openreview.net/forum?id=w4nd5695sq.

WSM$^+$25. Emily Wenger, Eshika Saxena, Mohamed Malhou, Ellie Thieu, and Kristin Lauter. Benchmarking attacks on learning with errors. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 279–297, 2025. doi:10.1109/SP61157.2025.00058.

Wun19. Thomas Wunderer. A detailed analysis of the hybrid lattice-reduction and meet-in-the-middle attack. *Journal of Mathematical Cryptology*, 13(1):1–26, 2019. `doi:10.1515/jmc-2016-0044`.

YD17. Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 3–22. Springer, Cham, August 2017. `doi:10.1007/978-3-319-72565-9_1`.