Cool + Cruel = Dual, and New Benchmarks for Sparse LWE

Alexander Karenin[1], Elena Kirshanova[1], Julian Nowakowski[2⋆], Eamonn W. Postlethwaite[3], Ludo N. Pulles[4⋆⋆], Fernando Virdia[5⋆⋆⋆], and Paul Vié[6†]

[1] Technology Innovation Institute, Abu Dhabi, United Arab Emirates
[2] Ruhr University Bochum, Bochum, Germany
[3] King's College London, London, United Kingdom
[4] Centrum Wiskunde & Informatica, Amsterdam, the Netherlands
[5] University of Surrey, Guildford, United Kingdom
[6] Télécom Paris, Paris, France

Abstract. The sparse secret Learning with Errors (LWE) problem is a widely used assumption in efficient fully homomorphic constructions. In [Wenger et al. IEEE S&P 2025] two approaches, 'Cool and Cruel' (C+C) and the machine learning based 'SALSA', were benchmarked against the well established primal attack on sparse secrets. The authors concluded that C+C outperforms SALSA and both outperform the primal attack. In this work we show that the apparently novel C+C is an instantiation of a known dual attack [Albrecht, EUROCRYPT 2017]. To argue this we introduce a framework for dimension reduction in the bounded distance decoding problem that may be of independent interest. Furthermore we prove that the C+C 'phenomenon' is an expression of the geometry of the well known Z-shape basis in q-ary lattices, despite claims to the contrary. We also show that a correctly parametrised primal attack outperforms C+C both in parameter regimes studied by Wenger et al. and in new parameter regimes. To support this claim, we provide an open source implementation of two variants of the primal attack that are relevant for sparse secret LWE: Drop+Solve [May–Silverman, CaLC 2001] and Guess+Verify [Albrecht et al. SAC 2019].

Keywords: Bounded Distance Decoding · Cryptanalysis · Sparse LWE

1 Introduction

Learning with errors (LWE). The (search) LWE problem is to find a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and a vector $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$ where $\mathbf{e} \in \mathbb{Z}^m$ is a relatively short vector. LWE underpins lattice based

[⋆] The research of Nowakowski was supported by DFG grant 465120249.
[⋆⋆] The research of Pulles was supported by ERC-StG 947821 (ARTICULATE).
[⋆⋆⋆] The research of Virdia was supported by UKRI grant EP/Y02432X/1.
[†] Part of this work was done while Vié was an intern at CWI.

cryptographic standards, hence it is crucial to understand its concrete hardness. For efficiency reasons in LWE based constructions $\mathbf{s}$ is also short. For example, modern FHE schemes use sparse binary or ternary secrets [CKKS17,GV23].

State of the art practical attacks on LWE exploit the fact that $(\mathbf{A}, \mathbf{b})$ implies an instance of the *Bounded Distance Decoding (BDD)* problem. This asks one to find the vector $\mathbf{v}$ in a lattice Λ closest to a given target $\mathbf{t}$, with the promise that such $\mathbf{t}$ is much closer to $\mathbf{v}$ than to any other lattice vector. We write $\mathbf{t} = \mathbf{v} + \mathbf{x}$ for a short $\mathbf{x}$. To see LWE as a BDD problem, consider the lattice

$$\Lambda_{\text{LWE}} = \{(\mathbf{y}, \mathbf{z})^T \in \mathbb{Z}^m \times \mathbb{Z}^n \mid \mathbf{y} = \mathbf{A}\mathbf{z} \mod q\}.$$

Note that $\mathbf{v} = (\mathbf{A}\mathbf{s}, \mathbf{s}) = (\mathbf{b} - \mathbf{e}, \mathbf{s})$ is in Λ_{LWE} and, moreover, is close to the target $\mathbf{t} = (\mathbf{b}, \mathbf{0})$ thanks to the smallness of $\mathbf{s}$ and $\mathbf{e}$, explicitly $\mathbf{x} = (\mathbf{e}, -\mathbf{s})$.

Solving BDD. Two approaches to solve BDD (and therefore LWE) are the primal and dual attacks. The primal attack, also known as Kannan's embedding, takes the target $\mathbf{t}$, the lattice basis $\mathbf{B}$ and constructs the following the lattice basis

$$\mathbf{B}' = \begin{pmatrix} \mathbf{B} & \mathbf{t} \\ \mathbf{0} & \alpha \end{pmatrix} \in \mathbb{Z}^{(m+n+1)\times(m+n+1)},$$

for some parameter $\alpha > 0$. For well chosen α the lattice generated by $\mathbf{B}'$ has a unique (to sign) shortest vector $(\mathbf{x}, \alpha)^T$, which can be found via lattice reduction [Kan83]. Extensions of this approach [How07,Ber23,KKMN25] combine guessing of a part of the BDD secret with lattice reduction on a smaller dimensional lattice.

Dual attacks [Alb17,EJK20,GJ21,MAT22,DP23b,CMST25] find many short vectors of $\Lambda^* = \{\mathbf{y} \in \text{span}(\Lambda) \mid \langle \mathbf{y}, \Lambda \rangle \subseteq \mathbb{Z}\}$, the lattice dual to Λ. These can be used to solve decisional BDD, which is to decide if a given $(\Lambda, \mathbf{t})$ forms a valid BDD instance. This decision BDD oracle is used to sparsify or lower the dimension of a BDD instance, in either case making it easier to solve.

The primal and dual attacks give baseline concrete security estimates for LWE based constructions since other LWE specific attacks, such as the combinatorial [ACF+15] or algebraic [AG11] attacks, are considered uncompetitive for real world LWE parameters. To solve small and medium sized LWE instances in practice, the primal attack and its variants [Dar13,Boc25] are used.

Recent attacks and their benchmarks on very *sparse secret LWE.* Since 2022, two apparently new approaches for solving very sparse LWE appeared: machine learning (ML) techniques [WCCL22,LSW+23,LWAZ+23,SWL+24], under the common name SALSA, and what we refer to as the Cool+Cruel (or C+C) attack from [NMW+24]. These apporoaches were benchmarked in [WSM+25] against known attacks including the primal. The article concluded *'Cool&Cruel is currently the best attack on our benchmark settings'*. Moreover, machine learning techniques, even though inferior to C+C, were reported to outperform the primal attack too, see [WSM+25, Table 9]. These benchmarks cover LWE instances with *extremely* sparse secrets (Hamming weight ranging between 11 and

25). Though this regime does not threaten existing LWE-based constructions, it comes close to the setting of efficient FHE schemes that utilise sparse secrets [CGGI20,CKKS17]. Hamming weights in $\{22, 24\}$ have been proposed for 128-bit secure constructions when combined with large dimensions [MHWW24].

1.1 Our Contributions

On the theoretical side, we explain C+C in the BDD framework and show it is an instantiation of a known dual attack on LWE. We also note that the models used in the SALSA approach are trained on the same reduced dual bases as C+C. The core difference lies in what is commonly known as the search-to-decision part of a dual attack: while C+C utilises a specific scoring function, SALSA adapts Regev's search-to-decision reduction [Reg09] by tailoring it to the output of their model. Our results may be summarised as

1. **Generalising dimension reduction to BDD.** Dimension reduction (a.k.a. enumeration) is the first step in many dual attacks on LWE [Alb17,EJK20]. It guesses part of the LWE secret $\mathbf{s}$ and uses the dual lattice to determine if the guess is correct. If $\mathbf{s}_1 \in \mathbb{Z}_q^k$ is the first k entries of $\mathbf{s}$, then
$$\Lambda'_{\text{LWE}} = \{(\mathbf{y}, \mathbf{z})^T \in \mathbb{Z}^m \times \mathbb{Z}^{n-k} \mid \mathbf{y} = \mathbf{A}'\mathbf{z} \mod q\},$$
forms a BDD instance (with an appropriate target) in a lower dimension of $m + n - k$, where $\mathbf{A}' \in \mathbb{Z}_q^{m \times (n-k)}$ consists of the last $n - k$ columns of $\mathbf{A}$. We show that dimension reduction is not specific to LWE by generalising it to any BDD instance. This result is complementary to [DP23b] which generalises the alternative sparsification approach to BDD.
2. **Cool+Cruel = Dual.** The more geometric BDD view of LWE, together with the dimension reduction generalisation, captures an existing dual attack [Alb17]. Phrasing C+C in the BDD framework shows it is an instantiation of [Alb17]. Moreover, contrary to a claim in [NMW$^+$24], we explain why the C+C 'phenomenon' is a consequence of a Z-shape basis.
3. **SALSA $\approx$ Dual+Regev's search-to-decision.** SALSA performs lattice reduction on the same dual lattices as C+C. The model is trained on these reduced dual bases and is then used to build a search-to-decision LWE oracle analogous to Regev's search-to-decision reduction [Reg09]. The model therefore appears to implicitly realise a dual attack based distinguisher.

Despite C+C not being conceptually new, one may still wonder whether in practice this version of the dual attack always obsoletes the primal attack. Consulting the benchmarks from [WSM$^+$25] would suggest this is the case.

On the practical side, we have two objectives: to demonstrate the primal attack is competitive with C+C, contrary to statements in [WSM$^+$25], and solving comparable instances to those in [WSM$^+$25]. Due to the differences in available hardware, we approach these as separate problems with data reported in Table 2.

To prove competitiveness, we run the primal attack on the same hardware as C+C. We do this by defining new sparse secret LWE instances in a similar but lower-dimensional regime to those in [WSM$^+$25], and solving them on the same machine using both the well-known Drop + Solve [MS01] (D+S) version of the primal attack and C+C. Our experiments suggest that D+S has competitive success probability with C+C within comparable wall time. For example, for an LWE instance with parameters $n = 128$ and $q = 2^{50}$, D+S succeeds on 10 out of 10 experiments, while C+C succeeds in 8 out 10 experiments. As part of these experiments, we give a new detailed analysis of the effect of rounding LWE samples, a technique that alleviates the need for high floating point precision when attacking instances in the $q \geq 2^{30}$ regime at the cost of increasing the noise-to-modulo ratio.

In order to solve the module-LWE instances proposed in [WSM$^+$25] on our hardware, we resort to using the Guess + Verify [ACW19] (G+V) variant of D+S which internally uses Babai's Nearest Plane algorithm to verify guesses on the secret coordinates. Using GPUs and batched Babai's Nearest Plane calls to handle several guesses at once, G+V solves the first binomial secret instance set by [WSM$^+$25] after *at most* 6 hours in 5/6 cases, on a server with one 96-core CPU and an H100 GPU. In comparison, C+C only succeeded in 2/10 cases, required *at least* 28.1 hours on a cluster with 161 CPU cores and 256 V100 GPUs. While it is unknown whether C+C can recover a secret with the higher weight $h = 12$ for the same LWE parameters, G+V can within 36.4 hours. That is, G+V requires fewer resources, runs faster and achieves better success probabilities than C+C. Moreover, it is understood for a given sparse secret LWE instance how to parametrise G+V or D+S to solve it with high probability. No such understanding is reported for C+C.

Artifacts. Open source of our implementation is found at:

– https://gitlab.com/fvirdia/cool-plus-cruel-equals-dual
– https://github.com/ludopulles/GPUPrimalHybrid

2 Preliminaries

Notations. We denote vectors by lowercase bold letters, and matrices by upper case bold letters. For a vector $\mathbf{x}$ we denote by $\|\mathbf{x}\|$ its Euclidean norm. By $\mathbf{A}^T$ we denote the transpose of matrix $\mathbf{A}$. We denote by $\langle \mathbf{v}, \mathbf{w} \rangle$ the inner product of $\mathbf{v}$ and $\mathbf{w}$. The standard basis of $\mathbb{R}^n$ is $\mathbf{e}_1, \ldots, \mathbf{e}_n$, e.g., $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$. For $n \in \mathbb{N}_+$ let $[n] = \{1, \ldots, n\}$. For $x \in \mathbb{R}$ and $p \in \mathbb{R}_+$ define $x \bmod^{\pm} p$ to be the representative of $x \bmod p$ lying in $[-p/2, p/2)$. In particular, if x and p are integer then so is $x \bmod^{\pm} p$. For $r \geq 0$ and $m \in \mathbb{N}$, $S^{m-1}(r) = \{\mathbf{x} \in \mathbb{R}^m \colon \|\mathbf{x}\| = r\}$.

Probability. Let Z be a finite set and $I = (a, b) \subset \mathbb{R}$ be a bounded interval. Then we denote the discrete uniform distribution over the set Z with $U(Z)$, and the continuous uniform distribution over the interval I with $U(I)$ or $U(a, b)$. We define $U_q = U(Z)$ for $Z = [-q/2, q/2) \cap \mathbb{Z}$ and $q \in \mathbb{N}_+$, the uniform distribution over the integers balanced modulo q. We denote expectations with $\mathbb{E}$ and variances with $\mathbb{V}$. Let $q \in \mathbb{N}_{\geq 2}$ then $\mathbb{V}[U_q] = (q^2 - 1)/12$.

The unbiased sample variance is an estimate for the variance of a random variable for which the exact variance is not known. For a random variable X and i.i.d. samples $x_1, \ldots, x_N \leftarrow X$ let $\bar{x} = (x_1 + \cdots + x_n)/n$ be the sample mean, then the estimate is given by $V_X = (N-1)^{-1} \sum\limits_{i=1}^{N} (x_i - \bar{x})^2$. Taking the square root of the above to estimate the standard deviation of X introduces bias, but is nonetheless used as a simple estimate, $S_X = \sqrt{V_X}$.

Lattices. An m dimensional lattice Λ is a discrete finitely generated subgroup of $\mathbb{R}^m$. A sublattice Λ' is a subgroup $\Lambda' \leq \Lambda$. It may be described by a basis $\mathbf{B} \in \mathbb{R}^{m \times n}$ of linearly independent columns whenever $\Lambda = \mathcal{L}(\mathbf{B}) = \{\mathbf{B} \cdot \mathbf{x} \colon \mathbf{x} \in \mathbb{Z}^n\}$. We say that $\mathbf{B}$ generates the lattice Λ. Such a lattice has rank n and is full rank whenever $n = m$.

The determinant of a lattice Λ is an invariant that can be computed given any basis as $\det(\Lambda) = \sqrt{\det(\mathbf{B}^T\mathbf{B})}$. It is invariant due to the (absolute) unit determinant of unimodular matrices. If Λ is full rank then $\det(\Lambda) = |\det(\mathbf{B})|$.

We number from zero. If $\mathbf{B} = (\mathbf{b}_0 \cdots \mathbf{b}_{n-1})$ then we define two families of projection operators on $\mathbb{R}^m$. For $k \in \{0\} \cup [n]$ denote by π_k the orthogonal projection onto $\operatorname{span}(\mathbf{b}_0, \ldots, \mathbf{b}_{k-1})$, and by $\pi_k^\perp$ the orthogonal projection onto $\operatorname{span}(\mathbf{b}_0, \cdots \mathbf{b}_{k-1})^\perp$, the orthogonal complement. Note that the dependence on $\mathbf{B}$ is usually omitted. If $\mathbf{B}$ is not clear from context, we write $\pi_{\mathbf{B},k}$. Note that $\pi_0 \colon \mathbb{R}^n \to \{\mathbf{0}\}$ and $\pi_0^\perp = \mathbb{1}_{\mathbb{R}^m}$. A projected lattice $\mathcal{L}(\mathbf{B}_{[k,\ell)})$ is generated by the basis $\mathbf{B}_{[k,\ell)} := (\pi_k^\perp(\mathbf{b}_k), \ldots, \pi_{\ell-1}^\perp(\mathbf{b}_{\ell-1}))$. Note that for $k \in [n]$, $\mathcal{L}(\mathbf{B}_{[0,k)})$ is a sublattice generated by the first k basis vectors of $\mathcal{L}(\mathbf{B})$.

The dual of a lattice Λ is $\Lambda^* = \{\mathbf{w} \in \operatorname{span}(\Lambda) \colon \langle \mathbf{w}, \Lambda \rangle \subseteq \mathbb{Z}\}$. Let $\mathbf{B}$ be a basis of Λ, then $\mathbf{D} = \mathbf{B} \cdot (\mathbf{B}^T\mathbf{B})^{-1}$ is a basis of Λ^*. If $\mathbf{B}$ is full rank, then $\mathbf{D} = (\mathbf{B}^T)^{-1}$. For all $k \in [n]$, $(\mathbf{b}_{n-1}, \ldots, \mathbf{b}_k)$ generate a lattice dual to $\mathbf{D}_{[k,d)} = (\pi_k^\perp(\mathbf{d}_k), \ldots, \pi_k^\perp(\mathbf{d}_{n-1}))$. Moreover, for $\mathbf{B}, \mathbf{D}$ as above, we call $(\mathbf{b}_{n-1}, \ldots, \mathbf{b}_0)$ a reversed dual basis of $\mathbf{D}$.

The first minimum of Λ is $\lambda_1(\Lambda) = \min_{\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}} \|\mathbf{v}\|$, the length of the shortest non zero lattice vector. The *Gaussian Heuristic* states that $\lambda_1(\Lambda)$ is approximated by $\mathrm{GH}(\Lambda) := \sqrt{n/2\pi e} \cdot \det(\Lambda)^{1/n}$.

Lattice reduction procedures are discussed throughout, where required with reference. Conceptually, all lattice reduction takes as input control parameters and a lattice basis $\mathbf{B}$, and outputs a basis $\mathbf{B}' = \mathbf{B} \cdot \mathbf{U}$ of better *quality.* Here quality is application specific and $\mathbf{U}$ is unimodular.

2.1 LWE and BDD

Definition 2.1 (Search LWE). *Let $n, m, q \in \mathbb{N}$. Let $\chi_\mathbf{s}$ and $\chi_\mathbf{e}$ be distributions on $\mathbb{Z}_q^n$ and $\mathbb{Z}^m$, respectively. The search* Learning with Errors Problem (LWE) *with parameters $(n, m, q, \chi_\mathbf{s}, \chi_\mathbf{e})$ is defined as follows. Given a uniformly* *random matrix* $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$*, and a vector* $\mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \mod q$*, where* $\mathbf{s} \leftarrow \chi_\mathbf{s}$ *and* $\mathbf{e} \leftarrow \chi_\mathbf{e}$*, output* $\mathbf{s}$*.*

Decision LWE asks one to distinguish LWE samples $(\mathbf{A}, \mathbf{b})$ from uniform over $\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. An algorithm solving decision LWE can be turned into an algorithm for Definition 2.1 in time $\mathrm{poly}(n, \log q)$, [MM11,MP12].

While LWE is fundamentally a lattice problem, its definition does not make this immediately explicit. We view LWE as the lattice problem that it truly is: an average case variant of the *Bounded Distance Decoding Problem (BDD).*

Bounded Distance Decoding (BDD). The BDD problem is defined as follows.

Definition 2.2 (BDD, primal formulation). *Given a basis* $\mathbf{B}$ *of some lattice* Λ*, and a* target $\mathbf{t}$*, where* $\mathbf{t} = \mathbf{v} + \mathbf{x}$*, for some* $\mathbf{v} \in \Lambda$ *and* $\mathbf{x}$ *with* $\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)$*, output* $\mathbf{v}$*.*

Note that the bound $\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)$ in Definition 2.2 provably guarantees uniqueness of the solution. That is, $\mathbf{v}$ will always be the closest lattice vector to $\mathbf{t}$. Under the Gaussian heuristic one can relax this bound to $\|\mathbf{x}\| < \lambda_1(\Lambda)$ and still expect a unique solution, see e.g. [DP23b, Sec. 2.2]. For $\|\mathbf{x}\| = r\lambda_1(\Lambda)$ with $r > 0$ and Λ rank d, the Gaussian heuristic predicts approximately r^d solutions.

Instead of requiring the output $\mathbf{v} \in \Lambda$ one may equivalently require the output of the error $\mathbf{x} = \mathbf{t} - \mathbf{v}$. In this case, BDD can be seen as the problem of finding the shortest representative of the equivalence class of $\mathbf{t}$ modulo Λ, with the same caveats about uniqueness if the restriction on $\|\mathbf{x}\|$ is weakened. Hence, BDD may be equivalently defined as follows.

Definition 2.3 (BDD, dual formulation). *Given a basis* $\mathbf{D}$ *of some dual lattice* Λ^**, and a* target $\mathbf{t}$*, where* $\mathbf{D}^T\mathbf{t} \equiv \mathbf{D}^T\mathbf{x} \mod 1$ *for some* $\mathbf{x}$ *with* $\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)$*, output* $\mathbf{x}$*.*

For the remainder of the paper we only consider Definition 2.3.

LWE as BDD. We show that a correctly parametrised LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$ is a BDD instance. Implicitly in the below when forming part of a basis or a target with an object over $\mathbb{Z}_q$ we consider the balanced lift to $\mathbb{Z}$. Form

$$\mathbf{D} := \frac{1}{cq}\begin{bmatrix} q\mathbf{I}_n & \mathbf{A}^T \\ & c\mathbf{I}_m \end{bmatrix}, \quad \text{and} \quad \mathbf{t} := \begin{bmatrix} 0^n \\ \mathbf{b} \end{bmatrix}, \tag{1}$$

where $c > 0$ is a parameter to be optimised. Then $\mathbf{D}$ is a basis of the dual Λ^* of the lattice

$$\Lambda := \mathcal{L}\left(\begin{bmatrix} c\mathbf{I}_n & \\ -\mathbf{A} & q\mathbf{I}_m \end{bmatrix}\right) = \left\{ \begin{bmatrix} c\mathbf{w} \\ \mathbf{y} \end{bmatrix} \in c\mathbb{Z}^n \times \mathbb{Z}^m \mid \mathbf{y} \equiv -\mathbf{A}\mathbf{w} \mod q \right\}. \tag{2}$$

Note, via $\mathbf{w} := -\mathbf{s}$ and $\mathbf{y} := \mathbf{b} - \mathbf{e} \equiv \mathbf{A}\mathbf{s} \mod q$, Λ contains the vector

$$\mathbf{v} := \begin{bmatrix} -c\mathbf{s} \\ \mathbf{b} - \mathbf{e} \end{bmatrix}. \tag{3}$$

When $(n, m, q, \chi_s, \chi_e)$ are chosen such that

$$\mathbf{x} := \begin{bmatrix} c\mathbf{s} \\ \mathbf{e} \end{bmatrix} = \mathbf{t} - \mathbf{v} \tag{4}$$

is likely shorter than $\frac{1}{2}\lambda_1(\Lambda)$, then $(\mathbf{D}, \mathbf{t})$ likely defines a BDD instance with solution $\mathbf{x}$.

Optimising c. BDD gets easier as the promise gets stronger, i.e. as $\mathbf{t}$ gets closer to Λ. In other words, the complexity of BDD grows with $\frac{\|\mathbf{x}\|}{\lambda_1(\Lambda)}$. By Eqs. (1) and (4) we have $\det(\Lambda) = c^n q^m$ and $\|\mathbf{x}\| = \sqrt{c^2\|\mathbf{s}\|^2 + \|\mathbf{e}\|^2}$. Hence, approximating $\lambda_1(\Lambda) \approx \mathrm{GH}(\Lambda)$ and viewing $\frac{\|\mathbf{x}\|}{\mathrm{GH}(\Lambda)}$ as a function of c, we have

$$\frac{\|\mathbf{x}\|}{\mathrm{GH}(\Lambda)} \propto \frac{\sqrt{c^2\|\mathbf{s}\|^2 + \|\mathbf{e}\|^2}}{c^{n/(n+m)}}. \tag{5}$$

The optimal choice for c to minimise the right hand side of Eq. (5) is

$$c = \frac{\|\mathbf{e}\|}{\sqrt{m}} \cdot \frac{\sqrt{n}}{\|\mathbf{s}\|}. \tag{6}$$

For this c we have $\|\mathbf{x}\| = \frac{\|\mathbf{e}\|}{\sqrt{m}}\sqrt{n+m}$, i.e. it balances the $n+m$ coordinates of $\mathbf{x}$ such that, on average, each coordinate is of size $\|\mathbf{e}\|/\sqrt{m}$. Choosing c *exactly* as in Eq. (6) is not always possible, as the norms $\|\mathbf{e}\|$ and $\|\mathbf{s}\|$ might be unknown a priori (or the optimal value might be irrational) However, this does not pose a significant issue in practice: often $\|\mathbf{s}\|$ and $\|\mathbf{e}\|$ are concentrated, for example on $\mathbb{E}[\|\chi_s\|]$ and $\mathbb{E}[\|\chi_e\|]$. Therefore, one may set

$$c = \left\lceil \frac{\mathbb{E}[\|\chi_e\|]}{\sqrt{m}} \cdot \frac{\sqrt{n}}{\mathbb{E}[\|\chi_s\|]} \right\rfloor.$$

3 Dual Attacks Revisited and Generalised

We start this section by defining the decision BDD problem and recalling existing methods to solve it that rely on short vectors from the dual lattice. Then we show a generalisation of the dimension reduction technique for LWE to BDD that enables a reduction from decision to search BDD.

3.1 Dual Distinguishing

While the literature often describes the dual attack as an attack against search LWE, its core is a distinguisher $\mathcal{D}_{\mathsf{dual}}$ that solves a *decision* version of BDD.

Definition 3.1 (Decision BDD). *Given a basis* $\mathbf{D}$ *of some dual lattice* Λ^**, and a target* $\mathbf{t}$*, decide if there exists an* $\mathbf{x}$ *such that* $\mathbf{D}^T\mathbf{t} \equiv \mathbf{D}^T\mathbf{x} \bmod 1$ *and* $\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)$.

Call an instance where such an $\mathbf{x}$ does exist a YES instance, and otherwise a NO instance. We write $\mathbf{t} = \mathbf{t}_\mathsf{Y}$ for targets in a YES instance and $\mathbf{t} = \mathbf{t}_\mathsf{N}$ otherwise.

To solve a decision BDD instance $(\mathbf{D}, \mathbf{t})$, the distinguisher $\mathcal{D}_\mathsf{dual}$ performs the following steps.

1. Compute a set $\mathcal{W}$ of *short* dual vectors $\mathcal{W} := \{\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}\} \subset \Lambda^*$, along with inner products $\mu_i := \langle \mathbf{w}_i, \mathbf{t} \rangle$ for all i.
2. Check whether the μ_i are sufficiently concentrated around the integers, if so decide YES ($\mathbf{t} = \mathbf{t}_\mathsf{Y}$). Otherwise, decide NO ($\mathbf{t} = \mathbf{t}_\mathsf{N}$).

Several realisations of the second step are given in [LW21, Sec. 4]. We give some intuition for $\mathcal{D}_\mathsf{dual}$. If $\mathbf{t} = \mathbf{t}_\mathsf{Y}$ then there exists $\mathbf{v}_\mathsf{Y} \in \Lambda$ and $\mathbf{x}_\mathsf{Y}$ with

$$\|\mathbf{x}_\mathsf{Y}\| = \frac{\gamma}{2}\lambda_1(\Lambda), \quad \text{where } \gamma < 1, \tag{7}$$

such that $\mathbf{t} = \mathbf{v}_\mathsf{Y} + \mathbf{x}_\mathsf{Y}$. If $\mathbf{t} = \mathbf{t}_\mathsf{N}$ then there exists $\mathbf{v}_\mathsf{N} \in \Lambda$ and $\|\mathbf{x}_\mathsf{N}\|$ with $\|\mathbf{x}_\mathsf{N}\| \geq \frac{1}{2}\lambda_1(\Lambda)$ such that $\mathbf{t} = \mathbf{v}_\mathsf{N} + \mathbf{x}_\mathsf{N}$. Hence, for any dual vector $\mathbf{w} \in \Lambda^*$,

$$\langle \mathbf{w}, \mathbf{t} \rangle \equiv \begin{cases} \langle \mathbf{w}, \mathbf{x}_\mathsf{Y} \rangle \bmod 1 & \text{if } \mathbf{t} = \mathbf{t}_\mathsf{Y}, \\ \langle \mathbf{w}, \mathbf{x}_\mathsf{N} \rangle \bmod 1 & \text{if } \mathbf{t} = \mathbf{t}_\mathsf{N}. \end{cases}$$

Let us model the distribution of $\mathbf{w}$ as uniform in a ball of some radius, as in [DP23a, Heuristic. 2]. Then the distributions of $\langle \mathbf{w}, \mathbf{x}_\mathsf{Y} \rangle$ and $\langle \mathbf{w}, \mathbf{x}_\mathsf{N} \rangle$ are close to Gaussian distributions with mean 0 and standard deviations

$$\sigma_\mathsf{Y} := \frac{1}{\sqrt{d}} \cdot \|\mathbf{x}_\mathsf{Y}\| \cdot \|\mathbf{w}\|, \quad \sigma_\mathsf{N} := \frac{1}{\sqrt{d}} \cdot \|\mathbf{x}_\mathsf{N}\| \cdot \|\mathbf{w}\|,$$

respectively [LW21, Section 5.1], where d is the dimension of Λ. (We caution the reader that more exact statements, that go beyond the requirements for this intuition, must be used for an accurate analysis [DP23a, Sec. 3.1].)

If $\mathbf{w}$ has particularly small norm $\|\mathbf{w}\| \leq \frac{1}{\gamma}\lambda_1(\Lambda^*)$, then by Eq. (7)

$$\sigma_\mathsf{Y} < \frac{1}{2\sqrt{d}} \cdot \lambda_1(\Lambda) \cdot \lambda_1(\Lambda^*).$$

For σ_N, it follows from $\|\mathbf{x}_\mathsf{N}\| \geq \frac{1}{2}\lambda_1(\Lambda)$ and $\|\mathbf{w}\| \geq \lambda_1(\Lambda^*)$ that

$$\sigma_\mathsf{N} \geq \frac{1}{2\sqrt{d}} \cdot \lambda_1(\Lambda) \cdot \lambda_1(\Lambda^*).$$

Hence, if $\mathbf{w}$ is sufficiently short then $\sigma_\mathsf{Y} < \sigma_\mathsf{N}$. In other words the distribution of $\langle \mathbf{w}, \mathbf{t}_\mathsf{Y} \rangle \bmod 1$ is concentrated more closely around 0 (equivalently: $\langle \mathbf{w}, \mathbf{t}_\mathsf{Y} \rangle$ around $\mathbb{Z}$) than the distribution of $\langle \mathbf{w}, \mathbf{t}_\mathsf{N} \rangle \bmod 1$. Intuitively, many short $\mathbf{w}$ distributed uniformly in some ball give a distinguisher.

If γ is close to 1 then there may be too few sufficiently short dual vectors for the uniform in a ball heuristic to be reasonable. Below we discuss some average case decision BDD instances for which γ is expected to be much smaller than 1.

Algorithm 1: Dual Distinguisher $\mathcal{D}_{\mathsf{dual}}$

```
Input: Decision BDD instance $(\mathbf{D}, \mathbf{t})$,
       short vectors $\mathcal{W} = \{\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}\} \subset \mathcal{L}(\mathbf{D})$, obtained from ShortVec
Output: Score $S \in [0, 1]$, indicating whether $\mathbf{t} = \mathbf{t}_{\mathsf{Y}}$ or $\mathbf{t} = \mathbf{t}_{\mathsf{N}}$
1 for $i = 0, \ldots, N-1$ do
2     $\mu_i := \langle \mathbf{w}_i, \mathbf{t} \rangle$.
3 return Score$(\mu_0, \ldots, \mu_{N-1})$
```

Average Case Decision BDD is discussed in the literature. In [LW21,DP23b] a YES instance has $\mathbf{x} \leftarrow U(S^{d-1}(r))$ and a NO instance has $\mathbf{x} \leftarrow U(\mathbb{R}^d/\Lambda)$. In [PS24] an LWE average case gives a YES instance as $\mathbf{x} \leftarrow \chi_e$ and a NO instance as $\mathbf{x} \leftarrow U(\mathbb{Z}^m/\Lambda_q(\mathbf{A}))$, where $\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{x} \bmod q \in \mathbf{A}\mathbb{Z}^n\}$. Recall Eqs. (2) and (4), then for the LWE instances of this article a YES instance has $\mathbf{x} \leftarrow c\chi_s \times \chi_e$ and a NO instance has $\mathbf{x} \leftarrow U((c\mathbb{Z}^n \times \mathbb{Z}^m)/\Lambda)$.

These average cases of decision BDD generate, with high probability, decision BDD instances with a large 'gap', i.e. $\|\mathbf{x}_{\mathsf{Y}}\|$ (resp. $\|\mathbf{x}_{\mathsf{N}}\|$) is much smaller (resp. greater) than $\frac{1}{2}\lambda_1(\Lambda)$. These instances are easier to distinguish than the worst case. Note that, like (search) BDD, these average cases only give decision BDD instances with some probability. For example, we may have that $\mathbf{x} \leftarrow c\chi_s \times \chi_e$ has $\Pr[\|\mathbf{x}\| \geq \frac{1}{2}\lambda_1(\Lambda)] > 0$. In addition, we always have that $\mathbf{x} \leftarrow U((c\mathbb{Z}^n \times \mathbb{Z}^m)/\Lambda)$ has $\Pr[\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)] > 0$, for example when $\mathbf{x} = \mathbf{0}$.

Instantiating $\mathcal{D}_{\mathsf{dual}}$. The literature suggests various techniques for instantiating the distinguisher $\mathcal{D}_{\mathsf{dual}}$. To obtain the set of short dual vectors $\mathcal{W} = \{\mathbf{w}_0, \ldots, \mathbf{w}_{N-1}\} \subset \Lambda^*$ one often begins by computing a BKZ reduced basis $\mathbf{D}_{\mathsf{BKZ}}$ of Λ^*, and then either

1. takes the basis vectors of $\mathbf{D}_{\mathsf{BKZ}}$ [NMW⁺24],
2. takes small linear combinations of the basis vectors of $\mathbf{D}_{\mathsf{BKZ}}$ [Alb17],
3. runs a discrete Gaussian sampler with a small standard deviation using $\mathbf{D}_{\mathsf{BKZ}}$ [PS24], or
4. runs a lattice sieve on some sublattice $\mathcal{L}(\mathbf{D}_{\mathsf{BKZ}[0:\beta)}) \subseteq \Lambda^*$, for some well chosen parameter $\beta \leq d$ [LW21,GJ21,MAT22,DP23b,CMST25].

In general, we assume black box access to an algorithm `ShortVec` that, on input a dual basis produces a list of short dual vectors $\mathbf{w}_0, \ldots, \mathbf{w}_{N-1} \in \Lambda^*$.

Regardless of the specific `ShortVec`, one decides whether $\mathbf{t} = \mathbf{t}_{\mathsf{Y}}$ or $\mathbf{t} = \mathbf{t}_{\mathsf{N}}$ via some suitable statistical test on the μ_i. Again, for the sake of generality, we assume black box access to some statistical test `Score`, that given the μ_i outputs a *score* $S \in [0, 1]$. Given some threshold, a score S close to 1 indicates that $\mathbf{t} = \mathbf{t}_{\mathsf{Y}}$ and otherwise that $\mathbf{t} = \mathbf{t}_{\mathsf{N}}$. The distinguisher $\mathcal{D}_{\mathsf{dual}}$ is given in Algorithm 1.

From Decision to Search. Modern dual attacks against LWE proceed in two steps. First, $\mathcal{D}_{\mathsf{dual}}$ is used to transform the initial hard LWE instance into an easier one. After that, the resulting easier instance is solved.

Algorithm 2: Dimension Reduction Dual Attack

```
   Input: BDD instance $(\mathbf{D}, \mathbf{t})$ over $d$ dimensional lattice $\Lambda$,
          parameters $k \in [d]$ and $\gamma > 0$
   Output: Solution $\mathbf{x}$ of BDD instance $(\mathbf{D}, \mathbf{t})$
1  Compute the reversed dual basis $\mathbf{B}$ of $\mathbf{D}$.
2  Split $\mathbf{B}$ into two submatrices $\mathbf{B} = [\mathbf{B}_0, \mathbf{B}_1]$ with $\mathbf{B}_0 = \mathbf{B}_{[0,d-k)}$.
3  $S_{\mathsf{max}} := 0,\ \mathbf{u}_{\mathsf{max}} := \perp,\ \mathbf{t}_{\mathsf{max}} := \perp$
4  $\mathcal{W} := \texttt{ShortVec}(\texttt{BKZ}(\mathbf{D}_{[k,d)}))$
5  foreach $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 \in \mathcal{L}(\mathbf{B}_{[d-k,d)})$ with $\|\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 - \pi^{\perp}_{d-k}(\mathbf{t})\| \leq \gamma$ do
6      $\mathbf{t}_{\widetilde{\mathbf{u}}_1} := \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1)$
7      $S := \mathcal{D}_{\mathsf{dual}}(\mathbf{D}_{[k,d)}, \mathbf{t}_{\widetilde{\mathbf{u}}_1}, \mathcal{W})$
8      if $S \geq S_{\mathsf{max}}$ then
9          $S_{\mathsf{max}} := S,\ \mathbf{u}_{\mathsf{max}} := \widetilde{\mathbf{u}}_1,\ \mathbf{t}_{\mathsf{max}} := \mathbf{t}_{\widetilde{\mathbf{u}}_1}$
10 Compute a closest lattice vector $\mathbf{B}_0\widetilde{\mathbf{u}}_0 \in \mathcal{L}(\mathbf{B}_0)$ to $\mathbf{t}_{\mathsf{max}}$.
11 return $\mathbf{t} - \mathbf{B}_0\widetilde{\mathbf{u}}_0 - \mathbf{B}_1\mathbf{u}_{\mathsf{max}}$.
```

To implement the first step, the literature suggests two techniques: *dimension reduction* and *sparsification*. In the LWE literature, the former is sometimes called the *enumeration part*, whereas the latter is called the *FFT part* (see [DP23b, Section 3.1] for why the FFT part is best understood as a sparsification technique). As their names suggest, the dimension reduction and sparsification techniques transform the initial BDD instance that underlies LWE into a new BDD instance defined over a lower dimensional or a sparser lattice.

The sparsification technique was recently analysed in detail by Ducas and Pulles [DP23b]. While previous works described the sparsification technique in terms of LWE, [DP23b] showed that it can be applied more generally to all BDD instances. As our first result, we show now that the dimension reduction technique is also not specific to LWE, but can be generalised to all BDD instances.

3.2 Dimension Reduction Generalised

Let $(\mathbf{D}, \mathbf{t})$ be a BDD instance over some d dimensional lattice Λ, and let $\mathbf{B}$ be the reversed dual basis of $\mathbf{D}$. For some parameter $k \in [d]$ the dimension reduction technique in the dual attack tries to transform $(\mathbf{D}, \mathbf{t})$ into an easier BDD instance over the $(d-k)$ dimensional sublattice $\mathcal{L}(\mathbf{B}_{[0,d-k)}) \subset \Lambda$, which is then solved. The procedure is described in Algorithm 2, which we go on to explain.

Following Line 2, let $\mathbf{B} = [\mathbf{B}_0, \mathbf{B}_1]$ with $\mathbf{B}_0 = \mathbf{B}_{[0,d-k)}$, i.e. split $\mathbf{B}$ into its first $d-k$ and last k columns. Let $\mathbf{x}$ be the solution to BDD instance $(\mathbf{D}, \mathbf{t})$. There exist $\mathbf{u}_0 \in \mathbb{Z}^{d-k}$ and $\mathbf{u}_1 \in \mathbb{Z}^k$ such that

$$\mathbf{t} = \mathbf{B}_0\mathbf{u}_0 + \mathbf{B}_1\mathbf{u}_1 + \mathbf{x}. \tag{8}$$

Applying $\pi^{\perp}_{d-k}(\cdot) = \pi^{\perp}_{\mathrm{span}(\mathbf{B}_0)}(\cdot)$ and $\pi_{d-k}(\cdot) = \pi_{\mathrm{span}(\mathbf{B}_0)}(\cdot)$ to Eq. (8), we obtain

$$\pi^{\perp}_{d-k}(\mathbf{t}) = \mathbf{B}_{[d-k,d)}\mathbf{u}_1 + \pi^{\perp}_{d-k}(\mathbf{x}), \tag{9}$$

and

$$\pi_{d-k}(\mathbf{t} - \mathbf{B}_1\mathbf{u}_1) = \mathbf{B}_{[0,d-k)}\mathbf{u}_0 + \pi_{d-k}(\mathbf{x}) = \mathbf{B}_0\mathbf{u}_0 + \pi_{d-k}(\mathbf{x}). \tag{10}$$

Using Eqs. (9) and (10), Algorithm 2 recovers $\mathbf{u}_0$ and $\mathbf{u}_1$ in a two step approach. Given $\mathbf{u}_0$ and $\mathbf{u}_1$ one immediately recovers the solution $\mathbf{x}$ via Eq. (8).

Step 1: Enumerating $\mathbf{u_1}$. For $\gamma > 0$ the for loop enumerates all vectors $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ with $\widetilde{\mathbf{u}}_1 \in \mathbb{Z}^k$ that lie within distance γ of the projected target $\pi^{\perp}_{d-k}(\mathbf{t})$. Suppose $\gamma \geq \|\pi^{\perp}_{d-k}(\mathbf{x})\|$. By Eq. (9), $\mathbf{u}_1$ is then among the enumerated $\widetilde{\mathbf{u}}_1$. Setting γ as close as possible to $\|\pi^{\perp}_{d-k}(\mathbf{x})\|$ minimises the number of enumerated $\widetilde{\mathbf{u}}_1$. Suppose $\|\mathbf{x}\|$ concentrates around some known value α, then $\|\pi^{\perp}_{d-k}(\mathbf{x})\| \approx \sqrt{k/d} \cdot \|\mathbf{x}\| \approx \sqrt{k/d} \cdot \alpha$. Hence, a good option in practice is to take γ slightly larger than $\sqrt{k/d} \cdot \alpha$.

Step 2: Identifying $\mathbf{u_1}$, solving for $\mathbf{u_0}$. For every $\widetilde{\mathbf{u}}_1$, Line 6 computes

$$\mathbf{t}_{\widetilde{\mathbf{u}}_1} := \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1). \tag{11}$$

Note that, by Eq. (10), if $\widetilde{\mathbf{u}}_1 = \mathbf{u}_1$ then

$$\mathbf{t}_{\mathbf{u}_1} = \mathbf{B}_{[0,d-k)}\mathbf{u}_0 + \pi_{d-k}(\mathbf{x}) = \mathbf{B}_0\mathbf{u}_0 + \pi_{d-k}(\mathbf{x}). \tag{12}$$

The crucial observation is that Eq. (12) defines a BDD instance $(\mathbf{D}_{[k,d)}, \mathbf{t}_{\mathbf{u}_1})$ on the lower dimensional lattice $\mathcal{L}(\mathbf{B}_{[0,d-k)}) = \mathcal{L}(\mathbf{D}_{[k,d)})^*$ with target $\mathbf{t}_{\mathbf{u}_1}$ and error $\pi_{d-k}(\mathbf{x})$. Since $\mathbf{x}$ is the error in the initial BDD instance over $\Lambda = \mathcal{L}(\mathbf{B})$, we have $\|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda)$. Moreover, since $\mathcal{L}(\mathbf{B}_{[0,d-k)}) = \mathcal{L}(\mathbf{B}_0) \subseteq \Lambda$, we have $\lambda_1(\Lambda) \leq \lambda_1(\mathcal{L}(\mathbf{B}_0))$. Hence,

$$\|\pi_{d-k}(\mathbf{x})\| \leq \|\mathbf{x}\| < \frac{1}{2}\lambda_1(\Lambda) \leq \frac{1}{2}\lambda_1(\mathcal{L}(\mathbf{B}_0)). \tag{13}$$

Thus, Eq. (12) is a well defined BDD instance. Line 7 tries to identify this BDD instance among the $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$ by using the dual distinguisher $\mathcal{D}_{\mathsf{dual}}$.

Recall $\mathcal{D}_{\mathsf{dual}}$ tries to solve decision BDD by assigning high scores to points close to the lattice, and low scores otherwise. Assume that $\mathbf{t}_{\mathbf{u}_1}$ receives the highest score among all $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$. Then, once the for loop terminates, Line 9 implies $\mathbf{u}_{\mathsf{max}} = \mathbf{u}_1$ and $\mathbf{t}_{\mathsf{max}} = \mathbf{t}_{\mathbf{u}_1}$, i.e. the for loop successfully identified $\mathbf{t}_{\mathbf{u}_1}$.

Line 10 then solves the lower dimensional BDD instance $(\mathbf{D}_{[k,d)}, \mathbf{t}_{\mathbf{u}_1})$ by computing the closest vector in $\mathcal{L}(\mathbf{B}_0)$ to $\mathbf{t}_{\mathbf{u}_1}$. This is $\mathbf{B}_0\mathbf{u}_0$ (Eqs. (12) and (13)), so it follows from Eq. (8) that the vector returned in Line 11,

$$\mathbf{t} - \mathbf{B}_0\mathbf{u}_0 - \mathbf{B}_1\mathbf{u}_1 = \mathbf{x},$$

is the solution to the initial BDD instance $(\mathbf{D}, \mathbf{t})$. Hence, if $(\mathbf{D}_{[k,d)}, \mathbf{t}_{\mathbf{u}_1})$ receives the highest score from $\mathcal{D}_{\mathsf{dual}}$ then Algorithm 2 successfully solves BDD.

Correctness. Ideally, Algorithm 2 would be instantiated with a $\mathcal{D}_{\mathsf{dual}}$ that is perfectly sensitive to distances between the $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$ and $\mathcal{L}(\mathbf{B}_0)$. That is,

$$\mathcal{D}_{\mathsf{dual}}(\mathbf{D}_{[k,d)}, \cdot, \mathcal{W}) \colon \mathbf{B}_0 \cdot \mathbb{R}^{d-k} \to [0,1]$$

strictly increases as $\operatorname{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}_0))$ decreases. Call such $\mathcal{D}_{\mathsf{dual}}$ *perfect*. For such perfect distinguishers, and exact knowledge of γ, Algorithm 2 is provably correct.

Theorem 3.2. *If Algorithm 2 is instantiated with a perfect distinguisher* $\mathcal{D}_{\mathsf{dual}}$ *and* $\gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|$ *then it returns the solution* $\mathbf{x}$ *of BDD instance* $(\mathbf{D}, \mathbf{t})$.

Proof. See Appendix A. □

Instantiating $\mathcal{D}_{\mathsf{dual}}$ and measuring its accuracy is ongoing research [DP23a,PS24] and beyond the scope of this article. A subtlety that is sometimes ignored is the probability that given many NO instances, the $\widetilde{\mathbf{u}}_1 \neq \mathbf{u}_1$ above, one may be closer to the lattice than the single YES instance [DP23b, Sec. 4.2]. The condition on γ in Theorem 3.2 ensures we avoid this contradictory regime, see Appendix A.

3.3 Implementing the For Loop for LWE

In general, to enumerate $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ with $\|\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 - \pi^{\perp}_{d-k}(\mathbf{t})\| \leq \gamma$ one may solve Approximate-CVP on lattice $\mathcal{L}(\mathbf{B}_{[d-k,d)})$ with target $\pi^{\perp}_{d-k}(\mathbf{t})$. For BDD instances derived from LWE there is a simpler approach.

Consider an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ and the BDD instance $(\mathbf{D}, \mathbf{t})$ derived from it, as in Eq. (1). For $k \leq n$, we have

$$\mathbf{D}_{[0,k)} = \frac{1}{qc}\begin{bmatrix} q\mathbf{I}_k \\ \mathbf{0}_{(d-k)\times k}\end{bmatrix},$$

and thus

$$\mathcal{L}(\mathbf{B}_{[d-k,d)}) = \mathcal{L}(\mathbf{D}_{[0,k)})^* = \left(\frac{1}{c}\mathbb{Z}^k \times \{0\}^{d-k}\right)^* = c\mathbb{Z}^k \times \{0\}^{d-k}.$$

Therefore, the for loop of Algorithm 2 on input $(\mathbf{D}, \mathbf{t})$ effectively enumerates over $c\mathbb{Z}^k$. Recall $\mathbf{t} = \mathbf{v} + \mathbf{x}$ such that $\mathbf{v}$ is the closest vector to $\mathbf{t}$ in $\mathcal{L}(\mathbf{B})$. The for loop enumerates the vectors $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ close to $\pi^{\perp}_{d-k}(\mathbf{t})$ to find the unique $\mathbf{u}_1 \in \mathbb{Z}^k$ with $\mathbf{B}_{[d-k,d)}\mathbf{u}_1 = \pi^{\perp}_{d-k}(\mathbf{v})$. In the LWE setting of Eq. (1), we have

$$\mathbf{v} = (-c\mathbf{s}, \mathbf{b} - \mathbf{e})^T \quad \text{and} \quad \mathbf{t} = (0^n, \mathbf{b})^T,$$

where $\mathbf{s}$ and $\mathbf{e}$ denote the LWE secret and error, see Eq. (3). Let $\mathbf{s}_1 \in \mathbb{Z}_q^k$ denote the first k coordinates of $\mathbf{s} \in \mathbb{Z}_q^n$. Since $\pi^{\perp}_{d-k}(\cdot)$ is the orthogonal projection onto $\operatorname{span}(\mathbf{B}_{[d-k,d)}) = \mathbb{R}^k \times \{0\}^{d-k}$, it follows that

$$\pi^{\perp}_{d-k}(\mathbf{v}) = (-c\mathbf{s}_1, 0^{d-k})^T \quad \text{and} \quad \pi^{\perp}_{d-k}(\mathbf{t}) = 0^d.$$

Enumerating vectors $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ that lie within distance γ of $\pi^{\perp}_{d-k}(\mathbf{t})$ is therefore equivalent to enumerating candidates $\widetilde{\mathbf{s}}_1 \in \mathbb{Z}^k$ for the first k coordinates of $\mathbf{s}$ with $\|c\widetilde{\mathbf{s}}_1\| \leq \gamma$. Depending on the secret distribution χ_s one may prefer to iterate through $\widetilde{\mathbf{s}}_1$ that are both possible according to χ_s and satisfy this length bound, and then use the implied $\widetilde{\mathbf{u}}_1$.

4 Instantiations: Dual Attacks on LWE

In this section we show that an existing dual attack falls under the framework of Algorithm 2. This enables us to further show C+C is also an instantiation of a dual attack. To see this we describe [Alb17] and C+C in our framework, noting that the latter is, except small inessential differences, a reformulation of the former. We also explain how the preprocessing step of machine learning approaches from [LWAZ+23,SWL+24] can be seen as an instantiation of a dual attack. Moreover, their distinguishing step is analogous to Regev's search-to-decision reduction tailored to their model and sparse secrets.

4.1 Albrecht's Dual Attack

Given an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$, Albrecht's dual attack splits $\mathbf{A} = [\mathbf{A}_u, \mathbf{A}_r] \in \mathbb{Z}_q^{m\times k} \in \mathbb{Z}_q^{m\times(n-k)}$ for some $k \in [n]$ and considers the lattice

$$\Lambda^{\mathsf{Alb}} := \left\{ \left(\frac{1}{c}\mathbf{y}, \mathbf{z}\right)^T \in \frac{1}{c}\mathbb{Z}^{n-k} \times \mathbb{Z}^m \mid \mathbf{y} \equiv \mathbf{A}_r^T \mathbf{z} \mod q \right\},$$

where c is some scaling factor.[7] Since Λ^{Alb} is generated by

$$\mathbf{D}^{\mathsf{Alb}} := \frac{1}{c} \begin{bmatrix} q\mathbf{I}_{n-k} & \mathbf{A}_r^T \\ \mathbf{0} & c\mathbf{I}_m \end{bmatrix},$$

it follows that, up to scaling by q, Λ^{Alb} is isomorphic to a projected sublattice of the lattice generated by our matrix $\mathbf{D}$ from Eq. (1). Indeed, letting $d = n + m$

$$\mathbf{D}_{[k,d)} = \frac{1}{cq} \begin{bmatrix} \mathbf{0}_{k\times n-k} & \mathbf{0}_{k\times m} \\ q\mathbf{I}_{n-k} & \mathbf{A}_r^T \\ \mathbf{0} & c\mathbf{I}_m \end{bmatrix} = \frac{1}{q} \begin{bmatrix} \mathbf{0}_{k\times(m+n-k)} \\ \mathbf{D}^{\mathsf{Alb}} \end{bmatrix}, \tag{14}$$

and we have

$$\Lambda^{\mathsf{Alb}} \simeq q \cdot \mathcal{L}(\mathbf{D}_{[k,d)}). \tag{15}$$

The attack computes a set $\mathcal{W}^{\mathsf{Alb}} \subset \Lambda^{\mathsf{Alb}}$ of short vectors. Using these vectors it then tries to determine the first k coordinates $\mathbf{s}_u$ of the LWE secret $\mathbf{s} = \mathbf{s}_u, \mathbf{s}_r)^T \in \mathbb{Z}_q^k \in \mathbb{Z}_q^{n-k}$. To this end, it iterates over all candidates $\widetilde{\mathbf{s}}_u$ for $\mathbf{s}_u$ and applies, for every $(\frac{1}{c}\mathbf{y}_i, \mathbf{z}_i) \in \mathcal{W}^{\mathsf{Alb}}$, some statistical test to $\langle \mathbf{z}_i, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u\rangle \mod q$. If the test determines the distributions of $\langle \mathbf{z}_i, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u\rangle$ are sufficiently close to some narrow Gaussian, the attack decides $\widetilde{\mathbf{s}}_u$ is the correct guess for $\mathbf{s}_u$. The idea is that for *short* $(\frac{1}{c}\mathbf{y}_i, \mathbf{z}_i)^T \equiv (\frac{1}{c}\mathbf{A}_r^T\mathbf{z}_i, \mathbf{z}_i)^T \mod q$, the inner product

$$\langle \mathbf{z}_i, \mathbf{b} - \mathbf{A}_u\mathbf{s}_u\rangle \equiv \mathbf{z}_i^T(\mathbf{A}\mathbf{s} + \mathbf{e} - \mathbf{A}_u\mathbf{s}_u) \equiv \mathbf{z}_i^T(\mathbf{A}_r\mathbf{s}_r + \mathbf{e}) \equiv \mathbf{y}_i^T\mathbf{s}_r + \mathbf{z}_i^T\mathbf{e} \mod q \tag{16}$$

[7] In [Alb17] Λ^{Alb} consists of vectors $\left(\mathbf{z}, \frac{1}{c}\mathbf{y}\right)^T$. Furthermore, [Alb17] does not use the *last* $(n-k)$ columns $\mathbf{A}_r$ of $\mathbf{A}$ to define Λ^{Alb}, but the *first* $(n-k)$ columns. We introduce these purely syntactical changes to unify notation.

approximately follows a narrow Gaussian distribution, whereas $\langle \mathbf{z}_i, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u\rangle \bmod q$ with $\widetilde{\mathbf{s}}_u \neq \mathbf{s}_u$ is expected to be somewhat uniform. This hopefully allows one to distinguish the correct guess from the incorrect ones. Given $\mathbf{s}_u$ the remaining coordinates $\mathbf{s}_r$ of $\mathbf{s}$ are recovered by solving a lower dimensional LWE instance $(\mathbf{A}_r, \mathbf{b}_r)$, where

$$\mathbf{b}_r := \mathbf{b} - \mathbf{A}_u\mathbf{s}_u \equiv \mathbf{A}_r\mathbf{s}_r + \mathbf{e} \mod q. \tag{17}$$

The BDD View. We show this attack is identical to Algorithm 2 applied to the BDD instance $(\mathbf{D}, \mathbf{t})$ defined by Eq. (1). Line references refer to Algorithm 2.

By Eq. (15), computing short vectors $\mathcal{W} \subset \mathcal{L}(\mathbf{D}_{[k,d)})$ in Line 4 is identical to computing short vectors $\mathcal{W}^{\mathsf{Alb}} \subset \Lambda^{\mathsf{Alb}}$. Next via Section 3.3, enumerating lattice vectors $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ with $\|\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 - \pi^{\perp}_{d-k}(\mathbf{t})\| \leq \gamma$ in the for loop enumerates all candidates $\widetilde{\mathbf{s}}_u \in \mathbb{Z}^k$ for $\mathbf{s}_u$. More precisely, since every $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$ is of the form $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 = (-c\widetilde{\mathbf{s}}_u, 0^{d-k})^T$, provided that $\gamma \geq c\max\|\widetilde{\mathbf{s}}_u\|$, all possible candidates are enumerated.

For every $\widetilde{\mathbf{u}}_1$, Line 6 computes $\mathbf{t}_{\widetilde{\mathbf{u}}_1} = \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1)$. Here $\mathbf{B}_1$ is the last k columns of the reversed dual basis $\mathbf{B}$ of $\mathbf{D}$, and $\pi_{d-k}(\cdot)$ is the orthgonal projection onto the linear span of the first k columns $\mathbf{B}_0 = \mathbf{B}_{[0,d-k)}$ of $\mathbf{B}$. Since $\mathbf{B}$ is obtained by reversing the columns of

$$\mathbf{D}^{-T} = \begin{bmatrix} c\mathbf{I}_n & \mathbf{0} \\ -\mathbf{A} & q\mathbf{I}_m \end{bmatrix} = \begin{bmatrix} c\mathbf{I}_k & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & c\mathbf{I}_{n-k} & \mathbf{0} \\ -\mathbf{A}_u & -\mathbf{A}_r & q\mathbf{I}_m \end{bmatrix},$$

it follows that the matrices $\mathbf{B}_0$ and $\mathbf{B}_1$ are obtained by reversing the columns of

$$\begin{bmatrix} \mathbf{0}_{k\times(n-k)} & \mathbf{0}_{k\times m} \\ c\mathbf{I}_{n-k} & \mathbf{0}_{(n-k)\times m} \\ -\mathbf{A}_r & q\mathbf{I}_m \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} c\mathbf{I}_k \\ \mathbf{0}_{(n-k)\times k} \\ -\mathbf{A}_u \end{bmatrix}.$$

Together with $(-c\widetilde{\mathbf{s}}_u, 0^{d-k})^T = \mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 = \pi^{\perp}_{d-k}(\mathbf{B}_1)\widetilde{\mathbf{u}}_1$, this shows that

$$\mathbf{B}_1\widetilde{\mathbf{u}}_1 = (-c\widetilde{\mathbf{s}}_u, 0^{n-k}, -\mathbf{A}_u\widetilde{\mathbf{s}}_u)^T,$$

and thus

$$\mathbf{t}_{\widetilde{\mathbf{u}}_1} = \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1) = (0^n, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u)^T. \tag{18}$$

Line 7 uses $\mathcal{D}_{\mathsf{dual}}$ and short vectors $\mathcal{W}$ to score each $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$. Internally $\mathcal{D}_{\mathsf{dual}}$ computes inner products $\mu_i := \langle \mathbf{w}_i, \mathbf{t}_{\widetilde{\mathbf{u}}_1}\rangle$ for each $\mathbf{w}_i \in \mathcal{W}$. Since $\mathcal{W} \subset \mathcal{L}(\mathbf{D}_{[k,d)})$, it follows from Eq. (14) that every $\mathbf{w}_i$ is of the form

$$\mathbf{w}_i = \frac{1}{q}\left(0^k, \frac{1}{c}\mathbf{y}_i, \mathbf{z}_i\right)^T \in \{0\}^k \times \frac{1}{cq}\mathbb{Z}^{n-k} \times \frac{1}{q}\mathbb{Z}^m.$$

As a result, the μ_i are computed as

$$\mu_i = \langle \mathbf{w}_i, \mathbf{t}_{\widetilde{\mathbf{u}}_1}\rangle \equiv \frac{1}{q}\langle \mathbf{z}_i, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u\rangle \mod 1.$$

Scaling up by q, these are identical to the inner products in Eq. (16).

As in Albrecht's attack, Algorithm 2 applies a statistical test to the μ_i to identify $(-c\mathbf{s}_u, 0^{d-k})^T = \mathbf{B}_{[d-k,d)}\mathbf{u}_1$ among the candidates $(-c\widetilde{\mathbf{s}}_u, 0^{d-k})^T = \mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1$. Given $\mathbf{s}_u$, Line 9 sets $\mathbf{t}_{\mathsf{max}} := \mathbf{t}_{\mathbf{u}_1} = (0^n, \mathbf{b} - \mathbf{A}_u\mathbf{s}_u)^T$, see Eq. (18). Finally, Algorithm 2 solves the initial BDD instance $(\mathbf{D}, \mathbf{t})$ by solving the lower dimensional BDD instance $(\mathbf{D}_{[k,d)}, \mathbf{t}_{\mathsf{max}})$. By Eqs. (14) and (18), this is identical to solving the lower dimensional LWE instance defined by Eq. (17).

4.2 Cool + Cruel

We give pseudocode for C+C in Algorithm 3. Certain details are omitted as they are independent of our discussion. Briefly,

- a parameter ρ determines some notion of reduction quality, abstractly it is a function of the output of some lattice reduction procedure,
- a scaling parameter c that they call the 'penalty parameter' forms part of their dual bases,
- a repetition parameter N determines the number of reduction procedures required in total for their statistical test,
- a number n_u of entries of $\mathbf{s}$ to brute force is computed as some function of the output of the reduction procedures.

In the below we first focus on the output of a single iteration of the for loop in Algorithm 3, i.e. set $N = 1$. We therefore omit indices i. The functions $\mathsf{score{-}cruel}$ and $\mathsf{recover{-}cool}$ represent a statistical test and an algorithm that solves a lower dimensional LWE instance respectively.

Step 1: Lattice Reduction. Given an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$, the C+C attack considers the lattice Λ^{CC} spanned by the *columns*[8] of

$$\mathbf{D}^{\mathsf{CC}} := \begin{bmatrix} & c\mathbf{I}_m \\ q\mathbf{I}_n & \mathbf{A}^T \end{bmatrix}. \tag{19}$$

Up to scaling by cq and a permutation of coordinates, $\mathbf{D}^{\mathsf{CC}}$ generates the same lattice as $\mathbf{D}$ from Eq. (1). The C+C attack BKZ reduces (along with some custom reduction procedure) $\mathbf{D}^{\mathsf{CC}}$ to receive $\mathbf{D}^{\mathsf{red}} = \mathbf{D}^{\mathsf{CC}} \cdot \mathbf{U}$ for some $\mathbf{U} \in \mathrm{GL}(n+m, \mathbb{Z})$.

Step 2: Identify Cool and Cruel Bits. The C+C attack splits $\mathbf{U}$ as[9] $\mathbf{U} = \begin{bmatrix} \mathbf{U}_0 \\ \mathbf{U}_1 \end{bmatrix}$, with $\mathbf{U}_1 \in \mathbb{Z}^{m\times(n+m)}$, and computes

$$\mathbf{A}^{\mathsf{red}} := \mathbf{U}_1^T \cdot \mathbf{A} \mod q, \quad \mathbf{b}^{\mathsf{red}} := \mathbf{U}_1^T \cdot \mathbf{b} \mod q. \tag{20}$$

[8] [NMW+24] uses row notation, and thus works with the transpose of $\mathbf{D}^{\mathsf{CC}}$. Also note that in [NMW+24] c is called ω.

[9] In [NMW+24], $\mathbf{U}_0$ is called $\mathbf{C}$ and $\mathbf{U}_1$ is called $\mathbf{R}$.

Algorithm 3: Cool + Cruel [NMW+24]

```
Input: LWE search instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{4n \times n} \times \mathbb{Z}_q^{4n}$, secret distribution $\chi_s$,
       error distribution $\chi_e$, quality parameter $\rho > 0$
Output: Solution $\mathbf{s} \in \mathbb{Z}_q^n$ to LWE search instance $(\mathbf{A}, \mathbf{b})$
1  set $m = \lceil 7n/8 \rfloor$, $\mathbf{A}^{\mathbf{red}}$, $\mathbf{b}^{\mathbf{red}} = \emptyset$
2  set $N, c \in \mathbb{N}^+$ as functions of the input parameters
3  for $i \in [N]$ do
4      uniformly select a submatrix $\mathbf{A}_i \in \mathbb{Z}_q^{m \times n}$ of $\mathbf{A}$
5      select the corresponding $\mathbf{b}_i \in \mathbb{Z}_q^m$ of $\mathbf{b}$
6      let $\mathbf{D}_i = \begin{pmatrix} \mathbf{0} & c\mathbf{I}_m \\ q\mathbf{I}_n & \mathbf{A}_i^t \end{pmatrix} \in \mathbb{Z}^{(n+m)\times(n+m)}$ for any lift of $\mathbf{A}_i$ to $\mathbb{Z}^{m \times n}$
7      while reduction quality $\rho$ not achieved do
8          perform a lattice reduction procedure
9      let $\mathbf{U} \in \mathrm{GL}(n+m, \mathbb{Z})$ represent the lattice reduction
10     let $\mathbf{U}_1 \in \mathbb{Z}^{m \times (n+m)}$ be the bottom block of $\mathbf{U}$
11     append (vertically) $\mathbf{U}_1^T \mathbf{A}_i \in \mathbb{Z}_q^{(n+m)\times n}$ to $\mathbf{A}^{\mathbf{red}}$ and $\mathbf{U}_1^T \mathbf{b}_i \in \mathbb{Z}_q^{n+m}$ to $\mathbf{b}^{\mathbf{red}}$
12 compute integers $1 \le n_u, n_r \le n$ such that $n_u + n_r = n$ as a function of $\mathbf{A}^{\mathbf{red}}$
13 split $\mathbf{A}^{\mathbf{red}} = (\mathbf{A}_u^{\mathbf{red}}, \mathbf{A}_r^{\mathbf{red}}) \in \mathbb{Z}_q^{N(n+m)\times n_u} \times \mathbb{Z}_q^{N(n+m)\times n_r}$
14 describe $S \subseteq \mathbb{Z}_q^{n_u}$ of possible guesses for $\mathbf{s}_u$
15 for $\widetilde{\mathbf{s}}_u \in S$ in ascending Hamming weight and lexicographical order do
16     let $\mathbf{s}_u$ be the $\widetilde{\mathbf{s}}_u$ maximising $\mathsf{score{-}cruel}(\mathbf{b}^{\mathbf{red}} - \mathbf{A}_u^{\mathbf{red}} \widetilde{\mathbf{s}}_u)$
17 let $\mathbf{s}_r = \mathsf{recover{-}cool}(\mathbf{A}^{\mathbf{red}}, \mathbf{b}^{\mathbf{red}}, \mathbf{s}_u)$
18 if $\mathbf{s} = \mathbf{s}_u \| \mathbf{s}_r$ is incorrect with respect to $(\mathbf{A}, \mathbf{b})$ and $\chi_e$ then
19     return $\perp$
20 return $\mathbf{s}$
```

Let $\mathbf{A}_r^{\mathsf{red}} \in \mathbb{Z}_q^{(n+m)\times n_r}$ be the last n_r columns of $\mathbf{A}^{\mathsf{red}}$ for a well chosen $n_r \in [n]$. The central observation of [NMW+24] is that $\mathbf{A}_r^{\mathsf{red}}$ has entries with small sample standard deviation compared to U_q. In particular, the sample standard deviation is much smaller than $\sqrt{\mathbb{V}[U_q]} = \sqrt{(q^2-1)/12}$.

Conversely, letting $\mathbf{A}_u^{\mathsf{red}} \in \mathbb{Z}_q^{(n+m)\times n_u}$ be the first $n_u = n - n_r$ columns of $\mathbf{A}^{\mathsf{red}}$, [NMW+24] claims the sample standard deviation of entries should be $\sqrt{(q^2-1)/12}$. We show in Appendix B that this is not entirely correct. It is however true that the sample standard deviation of $\mathbf{A}_r^{\mathsf{red}}$ is significantly smaller than that of $\mathbf{A}_u^{\mathsf{red}}$. Based on this observation, the C+C attack calls the first n_u columns *cruel* and the remaining n_r columns *cool*. The respective entries of the secret are called the *cruel bits* and the *cool bits*. To choose n_u concretely the sample deviation of column entries is compared to some threshold.

Step 3: Secret Guessing. Now C+C splits secret $\mathbf{s}$ as $\mathbf{s} = (\mathbf{s}_u, \mathbf{s}_r)^T \in \mathbb{Z}_q^{n_u} \times \mathbb{Z}_q^{n_r}$ and tries to guess $\mathbf{s}_u$. To this end, C+C iterates over candidates $\widetilde{\mathbf{s}}_u$ for $\mathbf{s}_u$, and applies the statistical test $\mathsf{score{-}cruel}$ to each

$$\mathbf{b}^{\mathsf{red}} - \mathbf{A}_u^{\mathsf{red}} \cdot \widetilde{\mathbf{s}}_u \bmod q. \tag{21}$$

If $\mathsf{score{-}cruel}$ determines that the sample distribution of a guess $\mathbf{b}^{\mathsf{red}} - \mathbf{A}_u^{\mathsf{red}} \cdot \widetilde{\mathbf{s}}_u$ is sufficiently narrow then C+C decides $\widetilde{\mathbf{s}}_u = \mathbf{s}_u$. The idea being that

$$\mathbf{b}^{\mathsf{red}} - \mathbf{A}_u^{\mathsf{red}} \cdot \mathbf{s}_u \equiv \mathbf{U}_1^T(\mathbf{A}\mathbf{s} + \mathbf{e}) - \mathbf{A}_u^{\mathsf{red}} \cdot \mathbf{s}_u \equiv \mathbf{A}_r^{\mathsf{red}}\mathbf{s}_r + \mathbf{U}_1^T\mathbf{e} \mod q$$

should be concentrated around 0 mod q due to the small sample variance of $\mathbf{A}_r^{\mathsf{red}}$, the sparseness of $\mathbf{s}_r$, the shortness of $\mathbf{e}$ and a heuristic argument about the size of $\mathbf{U}_1$. Conversely, $\mathbf{b}^{\mathsf{red}} - \mathbf{A}_u^{\mathsf{red}} \cdot \widetilde{\mathbf{s}}_u$ with $\widetilde{\mathbf{s}}_u \neq \mathbf{s}_u$ is expected to be somewhat uniform. This hopefully allows one to distinguish the correct guess from the incorrect guesses. Once $\mathbf{s}_u$ is found $\mathbf{s}_r$ is recovered using $\mathsf{recover{-}cool}$, a simple procedure specified in [NMW+24, Alg. 1].

Understanding Step 2. Curiously, [NMW+24, App. A.3] stresses that the C+C phenomenon in Step 2 is 'distinct' from the well known Z-shape [How07, Sec. 3.1] of BKZ reduced bases of q-ary lattices, see also [AD21] and [DKL+18, App. C.3]. Unfortunately, [NMW+24] gives no justification for this claim, and it is in fact wrong.

The Z-shape is a phenomenon that occurs when BKZ reduces a q-ary lattice basis with $q \cdot \mathbf{e}_i$ as columns, e.g. the $q\mathbf{I}_n$ block in $\mathbf{D}^{\mathsf{CC}}$. In particular, there is some $\kappa \in [n]$ such that BKZ leaves the first κ columns of $\mathbf{D}^{\mathsf{CC}}$ unaltered, resulting in a reduced basis of the form

$$\mathbf{D}^{\mathsf{red}} = \begin{bmatrix} \mathbf{0}_{m\times\kappa} & \mathbf{D}_0 \\ q\mathbf{I}_\kappa & \mathbf{D}_1 \\ \mathbf{0}_{(n-\kappa)\times\kappa} & \mathbf{D}_2 \end{bmatrix}, \tag{22}$$

where $(\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2) \in \mathbb{Z}^{m\times(n+m-\kappa)} \times \mathbb{Z}^{\kappa\times(n+m-\kappa)} \times \mathbb{Z}^{(n-\kappa)\times(n+m-\kappa)}$. In fact [NMW+24, Fig. 5], which the authors claim shows that C+C is independent of the Z-shape, provides the first evidence that κ *is* the number of cruel bits, i.e. $\kappa = n_u$. Indeed, with blue lines representing post reduction quantities, the right subfigure gives $n_u \approx 150$ and the left subfigure shows $\kappa \approx n_u$ vectors of norm q. Further evidence is given in Appendix C.

Let $\mathbf{A} = [\mathbf{A}_u, \mathbf{A}_r] \in \mathbb{Z}_q^{m\times n_u} \times \mathbb{Z}_q^{m\times n_r}$. If $\kappa = n_u$ then, see e.g. [DDGR20, Remark 30], BKZ effectively runs only on the projected basis

$$\mathbf{D}^{\mathsf{CC}}_{[n_u, n+m)} = \begin{bmatrix} \mathbf{0}_{m\times n_r} & c\mathbf{I}_m \\ \mathbf{0}_{n_u\times n_r} & \mathbf{0}_{n_u\times m} \\ q\mathbf{I}_{n_r} & \mathbf{A}_r^T \end{bmatrix}, \tag{23}$$

and transforms it into

$$\mathbf{D}^{\mathsf{red}}_{[n_u, n+m)} = \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{0}_{n_u\times n_r} \\ \mathbf{D}_2 \end{bmatrix}. \tag{24}$$

A consequence of this is that most of the entries of $\mathbf{D}_0$ and $\mathbf{D}_2$ in $\mathbf{D}^{\mathsf{red}}$ are much smaller than those of $\mathbf{D}_1$; intuitively BKZ *considered* $\mathbf{D}_0$ and $\mathbf{D}_2$ but not $\mathbf{D}_1$.

More specifically, [DEP23] observed that the entries of $\mathbf{D}_1$ are close to uniform over $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$. As a result, we expect the sample standard deviation of the entries of $\mathbf{D}_1$ to be approximately $\sqrt{(q^2-1)/12}$. For entries of $\mathbf{D}_0$ and $\mathbf{D}_2$ that were acted upon by BKZ we expect the sample standard deviation to be much smaller. Using these observations, we may now compare the sample standard deviations of $\mathbf{A}_u^{\mathsf{red}}$ and $\mathbf{A}_r^{\mathsf{red}}$.

Since $\mathbf{D}^{\mathsf{red}} = \mathbf{D}^{\mathsf{CC}} \cdot \mathbf{U}$, it follows from Eqs. (19), (20) and (22) that

$$\begin{bmatrix} \mathbf{0}_{m\times n_u} & \mathbf{D}_0 \\ q\mathbf{I}_{n_u} & \mathbf{D}_1 \\ \mathbf{0}_{n_r\times n_u} & \mathbf{D}_2 \end{bmatrix} = \mathbf{D}^{\mathsf{CC}} \cdot \mathbf{U} = \begin{bmatrix} c\mathbf{U}_1 \\ q\mathbf{U}_0 + \mathbf{A}^T\mathbf{U}_1 \end{bmatrix} \equiv \begin{bmatrix} c\mathbf{U}_1 \\ (\mathbf{A}^{\mathsf{red}})^T \end{bmatrix} \mod q. \tag{25}$$

In particular, the first n_u rows of $\mathbf{A}^{\mathsf{red}} = (\mathbf{A}_u^{\mathsf{red}}, \mathbf{A}_r^{\mathsf{red}}) \in \mathbb{Z}_q^{(m+n)\times n_u} \times \mathbb{Z}_q^{(m+n)\times n_r}$ are zero and the remaining $m + n_r$ rows consist of $(\mathbf{D}_1^T, \mathbf{D}_2^T) \in \mathbb{Z}^{(m+n_r)\times n_u} \times \mathbb{Z}^{(m+n_r)\times n_r}$. It follows that a $n_u^2/(n_u(m+n)) = n_u/(m+n)$ fraction of the entries of $\mathbf{A}_u^{\mathsf{red}}$ consists of zeros. The remaining $(m+n_r)/(m+n)$ fraction consist of assumed uniform elements from $\mathbf{D}_1$. Hence, the entries of $\mathbf{A}_u^{\mathsf{red}}$ have sample standard deviation approximately

$$\sqrt{\frac{m+n_r}{m+n}\frac{q^2-1}{12}}.$$

Given that $\mathbf{A}_r^{\mathsf{red}}$ has the same proportion of zeros and is otherwise formed from the matrix $\mathbf{D}_2$ reduced by BKZ, its sample standard deviation is expected to be much smaller. The C+C phenomenon, i.e. the observation that the sample standard deviation of the first n_u columns $\mathbf{A}_u^{\mathsf{red}}$ is larger than that of the remaining n_r columns $\mathbf{A}_r^{\mathsf{red}}$, is thus nothing more than the phenomenon that in the Z-shape basis $\mathbf{D}^{\mathsf{red}}$ from Eq. (22), $\mathbf{D}_2$ has smaller entries than $\mathbf{D}_1$.

Cool + Cruel = Dual. We show that C+C is identical to Algorithm 2, and in particular a rephrased variant of Albrecht's dual attack. Specifically, C+C is equivalent to running Algorithm 2 with parameter $k = n_u$ on input $(\frac{1}{qc}\mathbf{D}^{\mathsf{CC}}, \mathbf{t}^{\mathsf{CC}})$, where $\mathbf{t}^{\mathsf{CC}} := (\mathbf{b}, 0^n)^T$. Additionally, the repetition parameter N denotes combining the output of $\mathcal{D}_{\mathsf{dual}}$ across N calls to Algorithm 2.

A minor difference between C+C and Algorithm 2 is the selection of k. In Algorithm 2 it is chosen as an input parameter, whereas in C+C it is chosen as a function of $\mathbf{A}^{\mathbf{red}}$. A simple modification to Algorithm 2, where k is chosen as a function of $\mathbf{D}$, possibly after some preprocessing, captures this distinction.

Let $\mathbf{A} = [\mathbf{A}_u, \mathbf{A}_r] \in \mathbb{Z}_q^{m\times n_u} \times \mathbb{Z}_q^{m\times n_r}$. In Section 4.1, when applying Algorithm 2 to the BDD instance $(\mathbf{D}, \mathbf{t})$ defined by Eq. (1), the for loop enumerates candidates $\widetilde{\mathbf{s}}_u \in \mathbb{Z}_q^k$ for the first k coordinates $\mathbf{s}_u$ of $\mathbf{s}$. For every $\widetilde{\mathbf{s}}_u$, Line 6 computes

$$\mathbf{t}_{\widetilde{\mathbf{u}}_1} := (0^n, \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u)^T,$$

see Eq. (18). $\mathbf{D}^{\mathsf{CC}}$ swaps the first n rows of $qc\mathbf{D}$ with the last m. Therefore, when running Algorithm 2 on input $(\frac{1}{qc}\mathbf{D}^{\mathsf{CC}}, \mathbf{t}^{\mathsf{CC}})$, the for loop computes

$$\mathbf{t}_{\widetilde{\mathbf{u}}_1} := (\mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u, 0^n)^T,$$

for each candidate $\widetilde{\mathbf{s}}_u$. Next, Line 7 runs $\mathcal{D}_{\mathsf{dual}}$ on input $(\frac{1}{qc}\mathbf{D}^{\mathsf{CC}}_{[k,d)}, \mathbf{t}_{\widetilde{\mathbf{u}}_1}, \mathcal{W})$, where $d = n + m$, and $\mathcal{W} \subset \frac{1}{qc}\mathcal{L}(\mathbf{D}^{\mathsf{CC}}_{[k,d)})$ is the output of `ShortVec` on Line 4.

Suppose $\mathcal{W}$ consists of the basis vectors of $\mathtt{BKZ}(\frac{1}{qc}\mathbf{D}^{\mathsf{CC}}_{[k,d)})$. Then $\mathcal{W}$ is the columns $\frac{1}{qc}\mathbf{D}^{\mathsf{red}}_{[k,d)}$, for $\mathbf{D}^{\mathsf{red}}_{[k,d)}$ given in Eq. (24). It follows that the inner products $\mu_i = \langle \mathbf{w}_i, \mathbf{t}_{\widetilde{\mathbf{u}}_1} \rangle$ for $\mathbf{w}_i \in \mathcal{W}$ computed in $\mathcal{D}_{\mathsf{dual}}$ satisfy

$$(\mu_1, \ldots, \mu_{|\mathcal{W}|})^T \equiv \frac{1}{qc}(\mathbf{D}^{\mathsf{red}}_{[n_u,d)})^T \begin{bmatrix} \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u \\ 0^n \end{bmatrix}$$
$$\equiv \frac{1}{qc}\begin{bmatrix} \mathbf{D}_0^T \; \mathbf{0}_{(m+n_r)\times n_u} \; \mathbf{D}_2^T \end{bmatrix} \begin{bmatrix} \mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u \\ 0^n \end{bmatrix} \equiv \frac{1}{qc}\mathbf{D}_0^T(\mathbf{b} - \mathbf{A}_u\widetilde{\mathbf{s}}_u) \mod 1.$$

Since, by Eq. (25), we have $c\mathbf{U}_1 \equiv \begin{bmatrix} \mathbf{0}_{m\times n_u} \; \mathbf{D}_0 \end{bmatrix} \bmod q$, it follows that the vector $\mathbf{b}^{\mathsf{red}} - \mathbf{A}^{\mathsf{red}}_u \cdot \widetilde{\mathbf{s}}_u \bmod q$ from Eq. (21) satisfies

$$\frac{1}{q}(\mathbf{b}^{\mathsf{red}} - \mathbf{A}^{\mathsf{red}}_u \cdot \widetilde{\mathbf{s}}_u) \equiv \frac{1}{q}\mathbf{U}_1^T(\mathbf{b} - \mathbf{A}_u \cdot \widetilde{\mathbf{s}}_u) \equiv \begin{bmatrix} \mathbf{0}_{m\times n_u} \; \frac{1}{qc}\mathbf{D}_0 \end{bmatrix}^T (\mathbf{b} - \mathbf{A}_u \cdot \widetilde{\mathbf{s}}_u)$$
$$\equiv (0^{n_u}, (\mu_1, \ldots, \mu_{|\mathcal{W}|}))^T \mod 1. \tag{26}$$

In particular the input $\mathbf{b}^{\mathsf{red}} - \mathbf{A}^{\mathsf{red}}_u\widetilde{\mathbf{s}}_u$ to score−cruel in Algorithm 3 is identical to, ignoring scaling and zeros, the input to $\mathcal{D}_{\mathsf{dual}}$ in Algorithm 2 for the particular `ShortVec` we describe. That is, C+C does nothing more than run a statistical test on the inner products μ_i, exactly as in Albrecht's dual attack.

The N repetitions of the for loop on Line 3 of Algorithm 3 represent N tuples of the form Eq. (26). Having $\mathbf{A}^{\mathsf{red}}$ be a vertical block matrix formed in Line 11 of Algorithm 3 is then equivalent to performing Algorithm 2 on the appropriate block diagonal, rank $N(n + m)$, instance of BDD. Explicitly, let $\mathbf{D}^{\mathsf{CC},(i)} = \mathbf{D}_i$, $\mathbf{b}^{(i)} = \mathbf{b}_i$ and $\mathbf{U}^{(i)} = \mathbf{U}$ from the for loop of Algorithm 3. Form the BDD instance $(\mathbf{D}, \mathbf{t})$ via the block diagonal sum $\mathbf{D} = \oplus_i \mathbf{D}^{\mathsf{CC},(i)}$ and $\mathbf{t} = (\mathbf{b}^{(1)}, 0^n, \ldots, \mathbf{b}^{(N)}, 0^n)$. Lattice reduction over $\mathbf{D}$ is given by the block diagonal sum $\mathbf{U} = \oplus_i \mathbf{U}^{(i)}$. Finally, $\mathcal{W}$ consists of the basis vectors of $\frac{1}{qc}\mathbf{D}^{\mathsf{CC},(i)}_{[k,d)}\mathbf{U}^{(i)}$ padded appropriately either side with zeros. This recovers N instances of Eq. (26) to compute the score over.

4.3 Distinguishing in SALSA

In a series of papers [WCCL22,LSW+23,LWAZ+23,SWL+24] machine learning and LWE samples with fixed secret $\mathbf{s}$ and error distribution χ_e are used to train a model $\mathcal{M}$. The model $\mathcal{M}$ approximates some unknown shift of the oracle $\mathsf{O}_{\mathbf{s},\chi_e} : \mathbb{Z}_q^n \to \mathbb{Z}_q$ such that $\mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a})$ samples $e \leftarrow \chi_e$ and returns $\mathbf{a}^T\mathbf{s} + e \bmod q$. That is, let $t \in \mathbb{Z}_q$ be arbitrary and unknown, then $\mathcal{M}(\mathbf{a})$ attempts to output b following $t + \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a})$.

If $\mathcal{M}$ approximates $t + \mathsf{O}_{\mathbf{s},\chi_e}$ 'well' then the papers attempt to solve search LWE using this implicit knowledge of $\mathbf{s}$. From [LWAZ+23] onwards only a distinguishing method is used, which we elaborate on below. How the LWE samples are used to train the machine learning model varies but from [LSW+23] onwards lattice reduction is applied to the basis Eq. (19), as in Cool + Cruel, with the

Algorithm 4: SALSA distinguisher for fixed weight binary secret

```
Input: model $\mathcal{M}$, Hamming weight $h$, secret dimension $n$, trial vectors
       $\{\mathbf{a}_j\}_{j=1}^{T} \subset \mathbb{Z}_q^n$
Output: guess $\mathbf{s}'$ for secret $\mathbf{s}$
1 set $C = [0, \ldots, 0]$ and $\mathbf{s}' = (0 \cdots 0)$ both of length $n$
2 for $i \in [n]$ do
3     for $j \in [T]$ do
4         $\ell \leftarrow U(\mathbb{Z}_q^n)$
5         $C[i] += |\mathcal{M}(\mathbf{a}_j) - \mathcal{M}(\mathbf{a}_j + \ell \mathbf{e}_i)|$   ▷ minimal representative mod $q$
6 let $i_1, \ldots, i_h$ be the indices of the $h$ largest entries of $C$
7 set the entries of $\mathbf{s}'$ indexed by $i_1, \ldots, i_h$ to one
8 return $\mathbf{s}'$
```

same stated aim of reducing the sample variance of entries. That is, the loop of Algorithm 3 ending line 11, up to minor details. Then $\mathbf{A}^{\mathsf{red}}$ and $\mathbf{b}^{\mathsf{red}}$ are input to a machine learning procedure which we consider as a black box.

The distinguishing method using $\mathcal{M}$ is similar in spirit to Regev's search to decision reduction [Reg09, Lem. 4.2] with restrictions on the secret and the caveat that $\mathcal{M}$ does not distinguish between distributions – it does not output a bit – but is rather a stronger resource from which an implicit distinguisher is created. Algorithm 4 considers χ_e equal to a uniform Hamming weight h binary secret, which can be adapted according to [LWAZ+23, Sec. 3] to handle ternary secrets. The intuition is that if $\mathcal{M}$ approximates $t + \mathsf{O}_{\mathbf{s},\chi_e}$ well then

$$\begin{aligned}\mathcal{M}(\mathbf{a}) - \mathcal{M}(\mathbf{a} + \ell \mathbf{e}_i) &\approx t + \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a}) - (t + \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a} + \ell \mathbf{e}_i)) \\ &= \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a}) - \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a} + \ell \mathbf{e}_i),\end{aligned}$$

which is small when $s_i = 0$ and large when $s_i = 1$. In particular, if $s_i = 0$ then $\mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a}) - \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a} + \ell \mathbf{e}_i)$ is distributed as $\chi_e - \chi_e$ and if $s_i = 1$ it is uniform. (Note here $\mathbf{e}_i$ is a canonical vector and not an error and s_i is an entry of $\mathbf{s}$.)

We briefly sketch two approaches that may recover Regev's search to decision reduction more directly. We make no claims that they would be more efficient or easier to realise. First one could define $\mathcal{M}$ as $\boldsymbol{\varepsilon}$-good for $\boldsymbol{\varepsilon} = (\varepsilon_{\mathbf{a}})_{\mathbf{a}}$ whenever there exists $t \in \mathbb{Z}_q$ such that the statistical distance $\Delta(\mathcal{M}(\mathbf{a}), t + \mathsf{O}_{\mathbf{s},\chi_e}(\mathbf{a})) \leq \varepsilon_{\mathbf{a}}$ for $\mathbf{a} \in \mathbb{Z}_q^n$. If $\|\boldsymbol{\varepsilon}\|_\infty$ is small and $s_i = 0$ then $\mathcal{M}(\mathbf{a} + \ell \mathbf{e}_i)$ for many i.i.d. uniform ℓ will be similar to $t + \mathbf{a}^T \mathbf{s} + \chi_e$, i.e. concentrated, else if $s_i = 1$ it will be similar to uniform. Second, one could train a model $\mathcal{M}'$ on secret $\mathbf{s}$ and error distribution χ_e such that $\mathcal{M}'(b, \mathbf{a}) \in \{0, 1\}$ *is* a distinguisher for uniform and LWE samples.

Finally, note the model $\mathcal{M}$ takes as input the same (up to lattice reduction procedures) information as the Cool + Cruel attack, i.e. reduced dual bases and associated samples, and is ultimately used to implicitly distinguish uniform and LWE distributions in order to solve search LWE. As such, it appears $\mathcal{M}$ is used to run a 'black box dual attack'. As Cool + Cruel outperforms the SALSA line of papers [WSM+25], we do not expand further on this idea.

5 Hardness Baseline: the Primal Attack

When estimating the hardness of solving an LWE instance the simplest competitive attack strategy to implement is the *primal attack*. It consists of constructing an integer lattice Λ containing an encoding of the LWE secret $\mathbf{s} \in \mathbb{Z}^n$ and error $\mathbf{e} \in \mathbb{Z}^m$ as its shortest vector (up to sign). Referring to Section 2.1, this coincides with seeing LWE as average case BDD, and building 'Kannan's embedding' [Kan87] to construct a lattice containing the BDD solution vector $\mathbf{x}$ as its shortest vector.

5.1 The Drop + Solve Attack

One can also choose to embed only part of $\mathbf{s}$ into $\mathbf{x}$, this is usually the optimal approach when χ_s is a sparse distribution, such as binary or ternary of fixed Hamming weight $h \ll n$. This is achieved by guessing which coordinates of $\mathbf{s}$ are non-zero, and constructing a new LWE instance where only the relevant coordinates of the $\mathbf{a}$ components of each sample are taken into consideration. When all non-zero coordinates are correctly guessed, the resulting LWE instance can be solved by lattice reduction. We describe this attack in more detail in Appendix F.1. We call this the 'Drop + Solve' attack, following the LWE-estimator.[10] We provide pseudocode for Drop + Solve in Algorithm 5 in Appendix F.3.

Cost estimates for this strategy for 'standard' parameters can be computed using estimator scripts such as the LWE-estimator, or its successor, the lattice-estimator.[11] These scripts operate by predicting the lattice reduction *block size* β required to successfully run the attack with high probability. Estimator script predictions against non-sparse Gaussian, binary and ternary distributions have been experimentally verified for the $\beta \geq 60$ regime [AGVW17], including refined predictions for the low success probability regime [PV21]. Notoriously, predicting costs for attacks depends on various heuristics on random q-ary lattices that only hold for large enough β, see [GN08, Sec. 4.2] and [CN11, Sec. 6.1]. LWE instances requiring such sufficiently large β define our use of 'standard' above.

In [WSM$^+$25], the authors benchmark a series of recently proposed attacks on sparse secret LWE, stating that C+C clearly outperforms the state of the art primal attack. Their analysis of the primal attack and their comparison methodology is severly flawed; we further expand on this in Appendix E. Crucially, they ignore the Drop + Solve variant.

It is not the first time that work investigating small-secret LWE ignores Drop + Solve in estimates and experiments [CCLS20]. In Appendix F.2, we describe the steps required to perform meaningful predictions of the cost of attacking these LWE instances using Drop + Solve. While estimates can be obtained by passing the appropriate parameters to estimator scripts,[12] we are not aware of

[10] https://bitbucket.org/malb/lwe-estimator/
[11] https://github.com/malb/lattice-estimator/
[12] We provide examples in our code release.

any experiments testing this trade-off. In Section 5.4, we examine LWE instances with sparsity and moduli similar to those considered in [WSM+25] and report on the largest dimensions for which we can successfuly solve them using Drop + Solve with hardware we have available. We also attempt to reproduce the attacks described in [WSM+25] (using their source code) on the same hardware and LWE instances for comparison.

Had Drop + Solve been taken into consideration, the resulting experiments and methodology would have made for a fair primal attack baseline to compare novel attacks against, as the steps correspond to the Drop + Solve estimation code present in the {LWE, lattice}-estimator scripts. However, there is still room to improve the basic Drop + Solve attack. In the case of FHE-like parameters, the large modulus q significantly slows down lattice reduction throughout. Furthermore, we notice that while q is very large, e.g. $q \geq 2^{26}$, the noise is relatively small, being sampled from a Gaussian distribution with standard deviation $\sigma_e = 3.19$. In Section 5.2, we explore how to exploit this gap, obtaining faster Drop + Solve attacks in practice, even if the resulting asymptotic cost may technically worsen.

5.2 Error vs. Modulus Trade-offs

Learning with rounding (LWR) is a hardness assumption related to LWE where, instead of outputting $(\mathbf{A}, \mathbf{A}\mathbf{s}+\mathbf{e})$, the error is introduced in the form of rounding. Given q and $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, one picks $p < q$ and defines the round-q-to-p function $\lfloor\cdot\rceil_p : x \mapsto \lfloor(p/q)\cdot x\rceil \bmod p$, extending it to vectors and matrices by applying it coefficientwise.[13] The LWR instance is $(\mathbf{A}, \lfloor\mathbf{A}\mathbf{s}\rceil_p)$. The rounding has introduced a deterministic 'rounding error' to $\mathbf{A}\mathbf{s}$.

In the case of $q \gg \sigma_e$ however, it can be noted that with high probability $\lfloor\mathbf{A}\mathbf{s}+\mathbf{e}\rceil_p = \lfloor\mathbf{A}\mathbf{s}\rceil_p$. While this is an interesting observation, it cannot be directly used to reduce the floating point precision required by lattice reduction, since the matrix $\mathbf{A}$ required to construct the primal attack embedding is still defined over the large ring $\mathbb{Z}_q$. Hence, we propose to also round down $\mathbf{A}$. The resulting LWE-like instance $\left(\lfloor\mathbf{A}\rceil_p, \lfloor\mathbf{b}\rceil_p\right) = \left(\lfloor\mathbf{A}\rceil_p, \lfloor\mathbf{A}\mathbf{s}+\mathbf{e}\rceil_p\right)$ is entirely defined over $\mathbb{Z}_p$, introducing a trade-off between the size of p and the noise introduced by the rounding, which we proceed to analyse.

Definition 5.1 (Centered Irwin—Hall distribution [Irw27,Hal27]). *Let $X_1, \dots, X_n$ be i.i.d. continuous random variables with $X_i \sim U(0,1)$. Let $Y = \sum_{i=1}^{n} X_i$. The Irwin–Hall distribution IH_n is the distribution of Y with density function f_{IH_n}. The* centred *Irwin–Hall distribution CIH_n has density function $f_{\mathsf{CIH}_n}(x) = f_{\mathsf{IH}_n}(x+n/2)$.*

Remark 5.2. The support of IH_n is $[0,n]$ and of CIH_n is $[-n/2, n/2]$. The expectation of IH_n is $n/2$ and of CIH_n is 0. The variance of both IH_n and CIH_n

[13] We observe that the specific representative of $x \bmod q$ does not affect the output mod p since $\lfloor(p/q)(x+q)\rceil = \lfloor(p/q)\cdot x + p\rceil = \lfloor(p/q)\cdot x\rceil + p = \lfloor(p/q)\cdot x\rceil \bmod p$.

is $n/12$. The centred Irwin–Hall distribution is the distribution of $Y - n/2 = \sum_{i=1}^{n}(X_i - 1/2)$, where $(X_i - 1/2) \sim U(-1/2, 1/2)$.

Definition 5.3 (Discrete centered Irwin-Hall distribution). *Let $n \geq 1$. We define the* discrete *centered Irwin-Hall distribution DCIH_n to be a variant of CIH_n restricted to the integers. DCIH_n has support $S = [-n/2, n/2] \cap \mathbb{Z}$, and mass function $f_{\mathsf{DCIH}_n}(x) = f_{\mathsf{CIH}_n}(x) / \sum_{y \in S} f_{\mathsf{CIH}_n}(y)$.*

Remark 5.4. In the regimes where we use the DCIH_n distribution, we observe that its variance equals $n/12$, the same as the variance of the CIH_n distribution.

Heuristic 5.1 *Let $(\mathbf{A}, \mathbf{b})$ be a sparse ternary secret LWE instance with modulus q, such that the secret vector $\mathbf{s}$ has h non-zero indices and the error $\mathbf{e}$ is sampled coordinatewise from a distribution with standard deviation σ_e. Define $\tilde{\mathbf{e}} = \lfloor \mathbf{b} \rceil_p - \lfloor \mathbf{A} \rceil_p \cdot s \bmod^{\pm} p$. If $p \cdot \sigma_e \ll q$ and $h + 1 < p$ then the m coefficients of $\tilde{\mathbf{e}}_i$ are distributed close to i.i.d. DCIH_{h+1}.*

Proof. See Appendix D.

We run experiments and verified that Heuristic 5.1 appears to hold at a variety of parametrisations. We report the results of our experiments in Appendix D.

Given the large values of q used in our Drop + Solve experiments, we can confidently round down to a more amenable value, say $p \approx 2^{10}$ or $p \approx 2^{15}$, while maintaining a relatively narrow error distribution thanks to the extremely sparse secret. Without a sparse secret, this trade-off does not appear viable. We note that asymptotic cost models do not capture the practical advantage of this trade-off: they usually disregard the polylog(q) cost of floating point arithmetic (which is especially heavy when it requires larger precision than natively available on the CPU); rather, with this rounding strategy such models would instead only see the noise-to-modulus ratio increasing, leading them to predict a harder attack.

5.3 The Guess + Verify Attack

Due to the limitations of our hardware, Drop + Solve could not break the larger MLWE instances solved via Cool + Cruel in [WSM+25] in a short time span. In this section, we propose using the Guess + Verify (G+V) attack introduced in [ACW19] to solve some of these larger instances.

Here, we explain how Guess + Verify can be made competitive against sparse secret LWE using further dimension reduction, cf. Section 4.1, GPUs and a batched variant of *Babai's Nearest Plane (NP) Algorithm* [Bab86]. Independent of the recovery part in the attack, we improve the lattice reduction part using BLASTER [DPS25]. Additional implementation details can be found in [PV25].

Further dimension reduction. Suppose we drop k coefficients $J \subset [n]$ from a secret $\mathbf{s}$, and guess the secret $\mathbf{s}$ has h' nonzero entries in $\mathbf{s}_J$, the k-dimensional vector with coefficients s_j for $j \in J$. Given a set of dropped columns J and $I = [n] \setminus J$, the Guess + Verify attack first performs BKZ-reduction on the basis

$$\begin{bmatrix} \mathbf{0} & \lfloor c \rceil \cdot \mathbf{I}_{n-k} \\ q\mathbf{I}_m & -\mathbf{A}_I \end{bmatrix}$$

giving reduced basis $\mathbf{B}'$. It then attempts to verify each possible guess $\widetilde{\mathbf{s}_J}$ of $\mathbf{s}_J$ by checking whether Babai's Nearest Plane outputs a short enough vector on input $(\mathbf{B}', \mathbf{b}' = \mathbf{b} - \mathbf{A}_J \widetilde{\mathbf{s}_J})$. Essentially, this verification step is a *primal* distinguisher solving a decision BDD problem (cf. Section 3) using primal basis $\mathbf{B}'$.

There is a large gap between the YES instance (correct guess) and NO instances (wrong guesses) of this decision BDD problem, so we can further reduce dimension by projecting orthogonally away from $\mathbf{B}'_{[0,d-d')}$ for some $d' < d$. Denoting $d = m + n - k$, we distinguish in the lower-dimensional BDD instance $\left(\mathbf{B}'_{[d-d',d)}, \pi^{\perp}_{d-d'}(\mathbf{b}')\right)$ for a certain *distinguishing dimension* $d' \leq d$, which needs to be sufficiently large for correctness and is desired to be small for speed. Reminiscent of the dual distinguishing issues [DP23b], one should pick the distinguishing dimension d' such that with probability 98%, say, hypothesis testing detects the correct guess while discarding all other wrong guesses.

Namely, first we heuristically assume $(\mathbf{e}, c\mathbf{s}_I)$ is normally distributed around $\mathbf{0}$ with same gaussian parameter σ_e as χ_e. Denote the cumulative density function of the chi-squared distribution $\chi^2_{d'}$ with parameter d' by $\varXi^2_{d'}$, and let $p_{\text{fn}} = p_{\text{fp}} = 0.01$. Then, d' is the smallest value satisfying the inequality

$$\sigma_e \sqrt{\varXi^2_{d'}(1 - p_{\text{fn}})} < \text{GH}(\mathbf{B}_{[d-d',d)}) \left(\frac{p_{\text{fp}}}{T}\right)^{1/d'},$$

where T equals the total number of wrong guesses. For this value of d', we then pick a norm bound B in between both sides of this inequality, to act as the primal distinguisher. A guess is considered correct if and only if the corresponding target has norm less than B after Babai's Nearest Plane. With probability at least $1 - p_{\text{fn}}$, the correct guess is detected. Heuristically, we assume a target is distributed uniformly in $\text{span}(\mathbf{B}_{[d-d',d)})/\mathcal{L}(\mathbf{B}_{[d-d',d)})$, so the expected number of incorrect guesses giving a target of norm less than B after Babai's Nearest Plane is at most p_{fp}. Markov's inequality then asserts all incorrect guesses are discarded with probability at least $1 - p_{\text{fp}}$.

Note that one can also guess the secret $\mathbf{s}_J$ has weight *at most* h' and that this sometimes represents a favourable time vs. success probability trade off. While the above method for determining B is technically invalidated in this case, as $(\mathbf{e}, c\mathbf{s}_I)$ needs to be modelled differently, success probability can only increase.

GPU implementation. We ported the (experimental) lattice reduction library BLASTER [DPS25] to the GPU, and refer to this by cuBLASTER. It uses the Python package `CuPy`, which allows the use of CUDA for linear algebra using a `NumPy` interface. All linear algebra and size reduction is performed on the GPU. Local LLL reduction is done on the CPU, as well as the local BKZ reduction, which performs pruned enumeration in this block. Since BLASTER is only able

Table 1: Hardware used for our experiments.

Name	CPU (quantity $\times$ 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)

to reduce lattice bases having small entries, we round the possibly irrational small-secret embedding constant c to an integer, making sure that the largest entry in the basis is q. To quickly enumerate all possible guesses $\mathbf{s}_J$ for one set J, we wrote some CUDA kernels in C++. To then verify the guesses, we ported a batched version of Babai's Nearest Plane [DPS25, App. A] to the GPU. This batched Babai's Nearest Plane is more efficient in practice because we can use a well-optimised library for the matrix multiplication.

Instantiating the attack. The attack parameters (k, m, β, h') are estimated using a variant of the lattice-estimator, if these are not given explicitly. Specifically, we estimate the cost of Guess + Verify by calling `LWE.primal_hybrid` using the parameters `mitm=False`, `babai=True` and `simulator=CN11`. Namely, in estimations, we use the Chen–Nguyen simulator [CN11], which accurately simulates BKZ 2.0 as implemented in `fplll`. The reduction quality of BLASTERBKZ is slightly different from BKZ 2.0. Therefore, we simulate multiple BKZ tours (8 or 24), although BLASTERBKZ performs one tour of progressive BKZ. For details, compare the function `CN11` in the file `estimator/simulator.py` that is part of the lattice-estimator hosted in our repository to the official lattice-estimator.[14]

5.4 Experimental Results

We proceed to run experiments in order to compare Cool + Cruel with Drop + Solve and Guess + Verify using the machines listed in Table 1. Performance of Guess + Verify is compared with the Cool + Cruel results reported in [WSM+25].

Drop + Solve instances. Following the methodology outlined in Section 5 and described in more detail in Appendix F, we set out to implement Drop + Solve and perform attacks on various LWE instances with sparse ternary secrets. Pseudocode is available in Appendix F.3. Our objective is to measure the performance of Drop + Solve to assess whether the primal attack is indeed not competitive with C+C, as asserted in [WSM+25]. We choose to explore a similar LWE instance space to [WSM+25], mindful of our more modest computational resources. In order to make the comparison between Drop + Solve and C+C easier we define a set of LWE instances, and attempt to solve them using our implementation of Drop + Solve and the [WSM+25] implementation of C+C.[15] Our LWE instances of choice are of moderate dimension ($128 \leq n \leq 384$), generally have

[14] https://github.com/malb/lattice-estimator, commit `352ddaf`.

[15] https://github.com/facebookresearch/LWE-benchmarking, commit `47e40e7`.

large moduli $q \geq 2^{30}$, have Gaussian errors with standard deviation $\sigma_e = 3.2$ and sparse ternary secrets with Hamming weights $h \in [8, 30]$. A summary of instances and attack results can be found in Table 2. A detailed description of how attack parameters were chosen can be found in Appendix G. These experiments are run across 94 cores on machine Y, using off-the-shelf FPLLL and Flatter [RH23] lattice reduction on the CPU.

Guess + Verify instances. We benchmark the Guess + Verify attack on instances chosen in [WSM+25], specifically the first two columns of the "Kyber MLWE" and "HE LWE" settings from [WSM+25, Table 9]. Because BLASTERBKZ works natively in these dimensions and with below 30 bit moduli, no modulus switching is necessary. Binomial instances ($\mathfrak{B}^2$) are Module-LWE instances of rank 2 over the ring $\mathbb{Z}[X]/(q, X^{256} + 1)$, while ternary instances are Ring-LWE instances using $\mathbb{Z}[X]/(q, X^{1024} + 1)$. In both cases, the prime q used in the instances equal the ones found in [WSM+25, Tables 5, 6]. We run G+V on the binomial instances with $q \approx 2^{28}$ by supposing the guessed secret part $\mathbf{s}_J$ has weight $h' \leq 3$, and on all other instances that $h' = 3$.

The attack script takes as input a number of 'workers'. Each worker independently guesses a set of columns J to drop, and attempts to verify the guess. It is recommended to take the number of workers equal to the smallest multiple of the number of GPUs such that GPU utilization is at 100%.

Results. Table 2 contains our experimental results.

Drop + Solve. Our first observation is that the Drop + Solve variant of the primal attack is competitive with C+C, contrary to reporting in [WSM+25]. Indeed, while our experiments are in the low block size regime they achieve an overall high success probability, using a number of guesses that is in line with the expected value $p_{\text{guess}}^{-1} = \binom{n}{k}/\binom{n-h}{k}$. We note that when both $k > 0$ and $k = 0$ are tested ($n = 128$), *i.e.* the primal attack with and without coordinates of $\mathbf{s}$ being dropped, choosing $k > 0$ provides a runtime improvement.

The case of C+C is mixed. We could only run the attack up to dimension 192 within reasonable time, and we observe a significant variance in the success probability of the attack. We also notice that in dimension $n = 128$ reducing the number of rows from the preprocessing matrices $\mathbf{A}_i$ seems to reduce the performance of C+C. On the other hand, this does not seem to be the case in dimension $n = 192$. Rather than trying to expand C+C experiments beyond $n = 192$, we opted to use our available GPU time on the Guess + Verify attack.

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

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, G+V solved a binomial instance of higher Hamming weight than achievable by C+C in [WSM+25, Table 9].

In addition, G+V's GPU-based recovery step is cheaper than C+C's. The total GPU utilization of C+C measured in GPU · wall hours is at least double that of G+V. C+C only uses GPUs for the recovery step, while G+V uses GPUs for lattice reduction and recovery, so the GPU utilization in G+V's recovery step is arguably lower than what is listed in Table 2. Moreover, only C+C's minimal GPU utilization is reported [WSM+25], so G+V'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, G+V's total GPU utilization (including lattice reduction) is still lower than that of C+C on every tested parameter set.

We remark that G+V showcases the lattice reduction library cuBLASter, which appears to be an order of magnitude faster than the combination of FPLLL, `flatter` [RH23] and `polish` [CLLT24] used by C+C. Now, G+V 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 2: Experimental comparison of Drop + Solve, Guess + Verify and Cool + Cruel against sparse-secret LWE and MLWE. We consider two secret/error distributions: $\chi_s = \mathfrak{B}^2$ (resp. Ter) corresponds to the so-called binomial (resp. ternary) setting as in [WSM+25, Table 5] (resp. [WSM+25, Table 6]). Column sets $(m, 7n/8)$ for C+C refer to different parametrisation of the attack as explained in Appendix G.2. 'Prec.' refers to the floating point precision used in FPLLL: `double` or `double double`. Processing of timings on MLWE parameters are discussed in Appendix G.4.

LWE parameters							**Drop + Solve**								**Cool + Cruel (from Section 5.4)**								
											avg.	avg.				succ.		avg. prep. (s)		avg. gues. (s)		avg. cruel	
χ_s	n	q	p	h	σ_e	m	prec.	k	β	succ.	time (s)	guesses	p_{guess}^{-1}	serv.	timeout	m	$7n/8$	m	$7n/8$	m	$7n/8$	m	$7n/8$
Ter	128	2^{10}	2^{10}	8	3.2	67	d	38	20	10/10	24.7	28.0	18.4	Y	27	0/10	0/10	0.0	0.0	0.0	0.0	0.0	0.0
Ter	128	2^{30}	2^{15}	20	3.2	35	d	0	40	9/10	96.5	1.0	1.0	Y	106	0/10	6/10	0.0	106.4	0.0	6.2	0.0	33.3
Ter	128	2^{30}	2^{15}	20	3.2	18	d	32	40	10/10	250.7	684.7	553.5	Y	540	0/10	9/10	0.0	540.4	0.0	49.4	0.0	33.0
Ter	128	2^{50}	2^{15}	20	3.2	35	d	0	40	9/10	99.6	1.0	1.0	Y	110	0/10	7/10	0.0	110.4	0.0	13.3	0.0	33.0
Ter	128	2^{50}	2^{15}	20	3.2	18	d	32	40	10/10	261.5	718.7	553.5	Y	477	0/10	8/10	0.0	477.4	0.0	56.4	0.0	33.0
Ter	192	2^{30}	2^{10}	8	3.2	104	dd	58	40	7/10	504.1	28.9	19.0	Y	555	8/10	6/10	555.4	555.2	24.6	27.6	108.8	109.5
Ter	192	2^{30}	2^{10}	10	3.2	100	dd	63	40	8/10	565.0	77.5	60.1	Y	724	3/10	1/10	724.2	724.2	23.4	0.7	108.7	109.0
Ter	192	2^{30}	2^{15}	8	3.2	92	dd	0	40	5/10	987.8	1.0	1.0	Y	1087	10/10	9/10	1087.4	1087.2	5.3	87.2	76.4	95.3
Ter	192	2^{30}	2^{15}	20	3.2	65	dd	34	40	10/10	684.9	81.5	62.1	Y	could not run								
Ter	192	2^{30}	2^{15}	30	3.2	105	dd	22	40	9/10	1541.6	83.9	53.5	Y	could not run								
Ter	256	2^{30}	2^{15}	8	3.2	157	dd	51	40	7/10	2303.5	10.9	6.1	Y	could not run								
Ter	256	2^{30}	2^{15}	10	3.2	151	dd	56	40	3/10	2226.8	5.3	12.4	Y	could not run								
Ter	256	2^{50}	2^{15}	8	3.2	157	dd	51	40	8/10	2244.4	12.4	6.1	Y	could not run								
Ter	256	2^{50}	2^{15}	10	3.2	151	dd	56	40	7/10	2137.4	9.3	12.4	Y	could not run								
Ter	384	2^{30}	2^{15}	8	3.2	155	dd	168	45	5/7	10986.9	180.2	152.0	Y	could not run								
Ter	384	2^{30}	2^{15}	8	3.2	150	dd	189	30	7/10	7270.1	396.3	243.0	Y	could not run								

MLWE parameters							**Guess + Verify**							**Cool + Cruel ([WSM+25, Tables 9, 11, 12])**		
											avg.	avg.			min.	min.
χ_s	n	q	p	h	σ_e	m	h'	k	β	succ.	core-hours	GPU-hours	serv.	succ.	core-hours	GPU-hours
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{12}$	q	11	1	72	3	393	46	5/6	≤ 523	6	Y	2/10	4508 ± 81 $(28 \cdot 161)$	26 ± 13 $(0.1 \cdot 256)$
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{12}$	q	12	1	72	3	395	46	4/5	≤ 2773	30	Y	could not run		
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{28}$	q	20	1	192	≤ 3	234	52	4/5	≤ 23	23	Z	3/10	1661 ± 76 $(11 \cdot 151)$	51 ± 13 $(0.2 \cdot 256)$
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{28}$	q	21	1	192	≤ 3	235	52	5/5	≤ 28	28	Z	3/10	1661 ± 76 $(11 \cdot 151)$	154 ± 13 $(0.6 \cdot 256)$
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{28}$	q	25	1	192	≤ 3	235	53	5/5	≤ 164	164	Z	1/10	1661 ± 76 $(11 \cdot 151)$	10752 ± 128 $(42 \cdot 256)$
Ter	$1 \cdot 1024$	$\approx 2^{26}$	q	11	3.19	197	3	768	49	5/5	≤ 833	10	Y	1/10	1856 ± 29 $(32 \cdot 58)$	102 ± 52 $(0.1 \cdot 1024)$
Ter	$1 \cdot 1024$	$\approx 2^{29}$	q	9	3.19	274	3	800	53	5/5	≤ 188	10	H	10/10	1833 ± 3 $(31.6 \cdot 58)$	31 ± 6 $(0.03 \cdot 1024)$
Ter	$1 \cdot 1024$	$\approx 2^{29}$	q	10	3.19	274	3	801	53	5/5	≤ 846	43	H	0/10	1833 ± 3 $(31.6 \cdot 58)$	no success

References

ACF$^+$15. Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *DCC*, 74(2):325–354, 2015. 2

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. 4, 23

AD21. Martin R. Albrecht and Léo Ducas. *Lattice Attacks on NTRU and LWE: A History of Refinements*, pages 15–40. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. 17

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016. 44

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 403–415, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 2

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. 21

Alb17. Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Cham, April / May 2017. 2, 3, 9, 13

AWHT16. Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 789–819. Springer, Berlin, Heidelberg, May 2016. 41

Bab86. László Babai. On Lovász'lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986. 23

Ber23. Daniel J Bernstein. Asymptotics of hybrid primal lattice attacks. *Cryptology ePrint Archive*, 2023. 2

Boc25. The Bochum Challenges Team. Bochum challenges, September 2025. Available at `https://bochum-challeng.es`. 2

CCLS20. Hao Chen, Lynn Chua, Kristin Lauter, and Yongsoo Song. On the concrete security of LWE with small secret. Cryptology ePrint Archive, Paper 2020/539, 2020. 21

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. 3

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. 2, 3

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. 27

CMST25. Kévin Carrier, Charles Meyer-Hilfiger, Yixin Shen, and Jean-Pierre Tillich. Assessing the impact of a variant of MATZOV's dual attack on kyber. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of *LNCS*, pages 444–476. Springer, Cham, August 2025. 2, 9

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. 21, 25

Dar13. TU Darmstadt. Learning with errors challenge, 2013. Available at https://www.latticechallenge.org/lwe_challenge/challenge.php. 2

DDGR20. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Cham, August 2020. 17, 41

DEP23. Léo Ducas, Thomas Espitau, and Eamonn W. Postlethwaite. Finding short integer solutions when the modulus is small. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 150–176. Springer, Cham, August 2023. 17

DKL$^+$18. Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018. 17

DP23a. Léo Ducas and Ludo N. Pulles. Accurate score prediction for dual-sieve attacks. Cryptology ePrint Archive, Report 2023/1850, 2023. 8, 12

DP23b. Léo Ducas and Ludo N. Pulles. Does the dual-sieve attack on learning with errors even work? In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 37–69. Springer, Cham, August 2023. 2, 3, 6, 9, 10, 12, 24, 33, 34

DPS25. Léo Ducas, Ludo N. Pulles, and Marc Stevens. Towards a modern LLL implementation. In *ASIACRYPT 2025*, 2025. To appear. 23, 24, 25

EJK20. Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a dual/hybrid approach to small secret LWE - A dual/enumeration technique for learning with errors and application to security estimates of FHE schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 440–462. Springer, Cham, December 2020. 2, 3

GJ21. Qian Guo and Thomas Johansson. Faster dual lattice attacks for solving LWE with applications to CRYSTALS. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 33–62. Springer, Cham, December 2021. 2, 9

GN08. Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, Berlin, Heidelberg, April 2008. 21

GV23. Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV Revisited. *Journal of Cryptology*, 36(2):12, 2023. 2

Hal27. P. Hall. The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika*, 19(3-4):240–244, 12 1927. 22

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. 2, 17

Irw27. J. O. Irwin. On the frequency distribution of the means of samples from a population having any law of frequency with finite moments, with special reference to pearson's type ii. *Biometrika*, 19(3-4):225–239, 12 1927. 22

Kan83. Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *15th ACM STOC*, pages 193–206. ACM Press, April 1983. 2

Kan87. Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12:415–440, 1987. 21

KKMN25. Alexander Karenin, Elena Kirshanova, Alexander May, and Julian Nowakowski. Fast slicer for batch-CVP: Making lattice hybrid attacks practical. Cryptology ePrint Archive, Paper 2025/1910, 2025. 2

LSW$^+$23. Cathy Yuanchen Li, Jana Sotáková, Emily Wenger, Mohamed Malhou, Evrard Garcelon, François Charton, and Kristin E. Lauter. SalsaPicante: A machine learning attack on LWE with binary secrets. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 2606–2620. ACM Press, November 2023. 2, 19

LW21. Thijs Laarhoven and Michael Walter. Dual lattice attacks for closest vector problems (with preprocessing). In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 478–502. Springer, Cham, May 2021. 8, 9

LWAZ$^+$23. Cathy Yuanchen Li, Emily Wenger, Zeyuan Allen-Zhu, Kristin Lauter, and Francois Charton. Salsa verde: a machine learning attack on learning with errors with sparse small secrets. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, 2023. 2, 13, 19, 20

MAT22. IDF MATZOV. Report on the security of lwe: improved dual lattice attack, 2022. https://zenodo.org/record/6412487. 2, 9

MHWW24. Shihe Ma, Tairong Huang, Anyu Wang, and Xiaoyun Wang. Accelerating BGV bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 403–432. Springer, Cham, May 2024. 3

MM11. Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Berlin, Heidelberg, August 2011. 6

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Berlin, Heidelberg, April 2012. 6

MS01. Alexander May and Joseph H Silverman. Dimension reduction methods for convolution modular lattices. In *International Cryptography and Lattices Conference*, pages 110–125. Springer, 2001. 4

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. 2, 3, 9, 15, 16, 17, 36, 44, 45

PS24. Amaury Pouly and Yixin Shen. Provable dual attacks on learning with errors. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 256–285. Springer, Cham, May 2024. 9, 12

PV21. Eamonn W. Postlethwaite and Fernando Virdia. On the success probability of solving unique SVP via BKZ. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 68–98. Springer, Cham, May 2021. 21, 41

PV25. Ludo N. Pulles and Paul Vié. Accelerating the primal hybrid attack against sparse LWE using GPUs. Cryptology ePrint Archive, Paper 2025/1990, 2025. 23

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009. 3, 20

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. 26, 27, 42

SWL$^+$24. Samuel Stevens, Emily Wenger, Cathy Yuanchen Li, Niklas Nolte, Eshika Saxena, Francois Charton, and Kristin Lauter. SALSA FRESCA: Angular embeddings and pre-training for ML attacks on learning with errors. Cryptology ePrint Archive, Report 2024/150, 2024. 2, 13, 19

WCCL22. Emily Wenger, Mingjie Chen, Francois Charton, and Kristin Lauter. SALSA: Attacking lattice cryptography with transformers. Cryptology ePrint Archive, Report 2022/935, 2022. 2, 19

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 58–58, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. 2, 3, 4, 20, 21, 22, 23, 25, 26, 27, 28, 40, 41, 42, 44, 46

– Appendix –

A Proof of Theorem 3.2 and discussion

In this section we prove Theorem 3.2, and compare it to the *contradictory regime* of [DP23b]. For better readability, let use restate Theorem 3.2.

Theorem A.1. *If Algorithm 2 is instantiated with a perfect distinguisher $\mathcal{D}_{\mathsf{dual}}$ and $\gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|$ then it returns the solution $\mathbf{x}$ of BDD instance $(\mathbf{D}, \mathbf{t})$.*

Proof. By the discussion in Section 3.2, we must show that $\mathcal{D}_{\mathsf{dual}}$ gives $(\mathbf{D}_{[k,d)}, \mathbf{t}_{\mathbf{u}_1})$ the highest score in Line 7, for $\mathbf{t}_{\mathbf{u}_1}$ as defined in Eq. (12). As $\mathcal{D}_{\mathsf{dual}}$ is a perfect distinguisher it suffices to show that for every $\widetilde{\mathbf{u}}_1 \neq \mathbf{u}_1$ enumerated by the for loop we have

$$\operatorname{dist}\left(\mathbf{t}_{\widetilde{\mathbf{u}}_1}, \mathcal{L}(\mathbf{B}_0)\right) > \operatorname{dist}\left(\mathbf{t}_{\mathbf{u}_1}, \mathcal{L}(\mathbf{B}_0)\right).$$

By Eqs. (12) and (13) we have $\operatorname{dist}\left(\mathbf{t}_{\mathbf{u}_1}, \mathcal{L}(\mathbf{B}_0)\right) = \|\pi_{d-k}(\mathbf{x})\|$. Thus, we must show for each $\widetilde{\mathbf{u}}_1 \neq \mathbf{u}_1$ and $\widetilde{\mathbf{u}}_0 \in \mathbb{Z}^{d-k}$ that

$$\|\mathbf{t}_{\widetilde{\mathbf{u}}_1} - \mathbf{B}_0\widetilde{\mathbf{u}}_0\| > \|\pi_{d-k}(\mathbf{x})\|. \tag{27}$$

Since $\mathbf{t}_{\widetilde{\mathbf{u}}_1} = \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1)$ (see Eq. (11)), we have

$$\mathbf{t}_{\widetilde{\mathbf{u}}_1} - \mathbf{B}_0\widetilde{\mathbf{u}}_0 = \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1) - \mathbf{B}_0\widetilde{\mathbf{u}}_0 = \pi_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1 - \mathbf{B}_0\widetilde{\mathbf{u}}_0).$$

Together with the Pythagorean theorem, this yields

$$\begin{aligned}\|\mathbf{t}_{\widetilde{\mathbf{u}}_1} - \mathbf{B}_0\widetilde{\mathbf{u}}_0\|^2 &= \|\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1 - \mathbf{B}_0\widetilde{\mathbf{u}}_0\|^2 - \|\pi^{\perp}_{d-k}(\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1 - \mathbf{B}_0\widetilde{\mathbf{u}}_0)\|^2 \\ &= \|\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1 - \mathbf{B}_0\widetilde{\mathbf{u}}_0\|^2 - \|\pi^{\perp}_{d-k}(\mathbf{t}) - \mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1\|^2. \end{aligned} \tag{28}$$

By definition, the $\widetilde{\mathbf{u}}_1$ enumerated by the for loop satisfy

$$\|\pi^{\perp}_{d-k}(\mathbf{t}) - \mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1\| \leq \gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|. \tag{29}$$

Moreover, since $\mathbf{t} - \mathbf{x} = \mathbf{B}_0\mathbf{u}_0 + \mathbf{B}_1\mathbf{u}_1$ (see Eq. (8)) is the *unique* closest lattice vector in $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}_0) + \mathcal{L}(\mathbf{B}_1)$ to our BDD target $\mathbf{t}$, it follows that for each $\widetilde{\mathbf{u}}_1 \neq \mathbf{u}_1$, we have

$$\|\mathbf{t} - \mathbf{B}_1\widetilde{\mathbf{u}}_1 - \mathbf{B}_0\widetilde{\mathbf{u}}_0\| > \|\mathbf{t} - \mathbf{B}_1\mathbf{u}_1 - \mathbf{B}_0\mathbf{u}_0\| = \|\mathbf{x}\|. \tag{30}$$

Plugging Eqs. (29) and (30) into Eq. (28), we obtain

$$\|\mathbf{t}_{\widetilde{\mathbf{u}}_1} - \mathbf{B}_0\widetilde{\mathbf{u}}_0\|^2 > \|\mathbf{x}\|^2 - \|\pi^{\perp}_{d-k}(\mathbf{x})\|^2 = \|\pi_{d-k}(\mathbf{x})\|^2,$$

which immediately implies Eq. (27), and thereby proves the theorem. □

Comparison with [DP23b]. Ducas and Pulles [DP23b] identified a contradictory regime in which dual attacks cannot work. Let T be the number of vectors $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 \in \mathcal{L}(\mathbf{B}_{[d-k,d)})$ enumerated in the for loop of Algorithm 2 and let

$$T^* := \left(\frac{\mathrm{GH}(\mathcal{L}(\mathbf{B}_{[0,d-k)}))}{\|\pi_{d-k}(\mathbf{x})\|}\right)^{d-k}. \tag{31}$$

For Algorithm 2, the contradictory regime is $T \geq T^*$. For such T, it is (heuristically) expected that one of the vectors $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$ computed in Line 6 has

$$\mathrm{dist}(\mathbf{t}_{\widetilde{\mathbf{u}}_1}, \mathcal{L}(\mathbf{B}_{[0,d-k)})) < \|\pi_{d-k}(\mathbf{x})\|,$$

see [DP23b, Heuristic Claim 4]. In particular, some $\mathbf{t}_{\widetilde{\mathbf{u}}_1}$ is expected to be closer to $\mathcal{L}(\mathbf{B}_{[0,d-k)})$ than the vector $\mathbf{t}_{\mathbf{u}_1} = \mathbf{B}_{[0,d-k)}\mathbf{u}_0 + \pi_{d-k}(\mathbf{x})$ from Eq. (12). As a result, $\mathbf{t}_{\widetilde{\mathbf{u}}_1} \neq \mathbf{t}_{\mathbf{u}_1}$ likely receives the highest score from $\mathcal{D}_{\mathsf{dual}}$ in Line 7, resulting in Algorithm 2 failing.

Let us compare the contradictory regime with Theorem 3.2, which shows that Algorithm 2 is *provably* correct when instantiated with $\gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|$ and a perfect distinguisher.

The for loop enumerates all $\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 \in \mathcal{L}(\mathbf{B}_{[d-k,d)})$ with $\|\mathbf{B}_{[d-k,d)}\widetilde{\mathbf{u}}_1 - \pi^{\perp}_{d-k}(\mathbf{t})\| \leq \gamma$. Hence, under the Gaussian heuristic we expect

$$T \approx \left(\frac{\gamma}{\mathrm{GH}(\mathcal{L}(\mathbf{B}_{[d-k,d)}))}\right)^{k}.$$

For $\gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|$ as in Theorem 3.2, we therefore have

$$T \approx \left(\frac{\|\pi^{\perp}_{d-k}(\mathbf{x})\|}{\mathrm{GH}(\mathcal{L}(\mathbf{B}_{[d-k,d)}))}\right)^{k} \approx \left(\sqrt{\frac{2\pi e}{d}}\|\mathbf{x}\|\right)^{k} \frac{1}{\det(\mathcal{L}(\mathbf{B}_{[d-k,d)}))},$$

where the second approximation follows from $\|\pi^{\perp}_{d-k}(\mathbf{x})\| \approx \sqrt{\frac{k}{d}}\|\mathbf{x}\|$. Noting

$$\|\mathbf{x}\| < \lambda_1(\mathcal{L}(\mathbf{B})) \approx \mathrm{GH}(\mathcal{L}(\mathbf{B})) = \sqrt{\frac{d}{2\pi e}}\det(\mathcal{L}(\mathbf{B}))^{1/d},$$

and treating approximations as equalities, it follows that the constraint $\gamma = \|\pi^{\perp}_{d-k}(\mathbf{x})\|$ in Theorem 3.2 thus ensures that

$$T < \frac{\det(\mathcal{L}(\mathbf{B}))^{k/d}}{\det(\mathcal{L}(\mathbf{B}_{[d-k,d)}))}. \tag{32}$$

Analogously to the above, one can show that T^* in Eq. (31) satisfies

$$T^* > \frac{\det(\mathcal{L}(\mathbf{B}_{[0,d-k)}))}{\det(\mathcal{L}(\mathbf{B}))^{(d-k)/d}}.$$

Together with

$$\frac{1}{\det(\mathcal{L}(\mathbf{B}))^{(d-k)/d}} = \frac{\det(\mathcal{L}(\mathbf{B}))^{k/d}}{\det(\mathcal{L}(\mathbf{B}))},$$

and

$$\det(\mathcal{L}(\mathbf{B}_{[0,d-k)})) = \frac{\det(\mathcal{L}(\mathbf{B}))}{\det(\mathbf{B}_{[d-k,d)})},$$

this shows

$$T^* > \frac{\det(\mathcal{L}(\mathbf{B}))^{k/d}}{\det(\mathcal{L}(\mathbf{B}_{[d-k,d)}))}. \tag{33}$$

Comparing Eqs. (32) and (33), it follows that $T < T^*$. Hence, the constraint $\gamma = \|\pi_{d-k}^{\perp}(\mathbf{x})\|$ in Theorem 3.2 ensures that Algorithm 2 avoids the contradictory regime.

B Inaccuracy of C+C's reported sample standard deviation

Recall that [NMW$^+$24] claims the sample standard deviation of the first few columns $\mathbf{A}_u^{\mathbf{red}}$ of $\mathbf{A}^{\mathsf{red}}$ (see Eq. (20)) equals that of uniform elements of $\mathbb{Z}_q$. This claim is inaccurate since the first n_u rows of $\mathbf{A}^{\mathsf{red}}$ are zero, i.e. very far from uniform, see Eq. (25).

This inaccuracy stems from [NMW$^+$24, Figures 1 and 5], which supposedly plot the sample standard deviations σ of the columns of $\mathbf{A}^{\mathsf{red}}$, misleadingly suggesting $\sigma \approx \sqrt{\frac{q^2-1}{12}}$ for $\mathbf{A}_u^{\mathbf{red}}$. Unfortunately, these plots were generated with an implementation of the C+C attack that differs from the description in the paper: after computing $\mathbf{A}^{\mathsf{red}} = \mathbf{U}_1^T \cdot \mathbf{A}$, the implementation calls a function `_make_RAs_RBs`,[16] removing every zero row from $\mathbf{A}^{\mathsf{red}}$. (This step is absent from the attack description of [NMW$^+$24].) By Eq. (25), `_make_RAs_RBs` thus removes the first n_u rows from $\mathbf{A}^{\mathsf{red}}$. As a result, the implementation uses $[\mathbf{D}_1^T, \mathbf{D}_2^T]$ in place of $\mathbf{A}^{\mathsf{red}}$. Since $\mathbf{D}_1$ is essentially uniformly random over $\mathbb{Z}_q$, this explains the inaccuracy of [NMW$^+$24, Figures 1 and 5].

[16] https://github.com/facebookresearch/cruel_and_cool/blob/main/data.py

C Z-shape determines cruel bits

In Fig. 1 we construct $\mathbf{D}^{\mathsf{CC}}$ as in Eq. (19) for $m = n = 128$ and growing q, then perform LLL reduction. The vertical red line indicates the final index where a q vector appears in $\mathbf{D}^{\mathbf{red}} = \mathbf{D}^{\mathsf{CC}}\mathbf{U}_{\mathsf{LLL}}$. Let σ be the sample standard deviation of some column of the resulting $\mathbf{A}^{\mathbf{red}}$ then the points indicate the normalised sample standard deviation of the column of the given index. High and low sample standard deviations (the C+C phenomenon) coincide exactly with the existence or absence of q vectors, i.e. columns $q \cdot \mathbf{e}_i$.

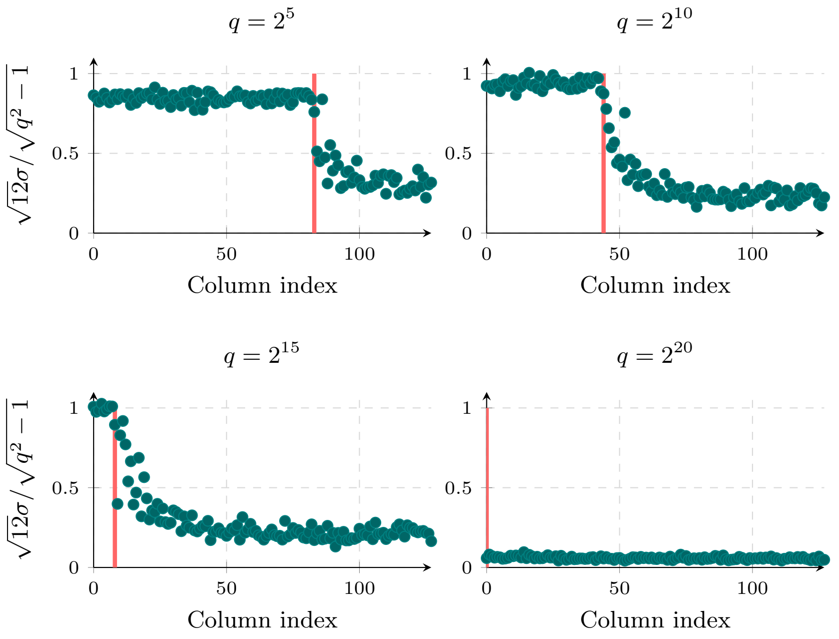


Fig. 1: Normalised sample standard deviation of entries within each column of $\mathbf{A}^{\mathsf{red}}$ for $n = m = 128$ and various q after LLL reduction. Vertical lines show the number of q vectors in the corresponding lattice basis $\mathbf{D}^{\mathsf{red}}$.

D Proof of Heuristic 5.1 and experiments on error vs. modulus trade-offs

Proof (heuristic of Heuristic 5.1). Let $\mathbf{A}_i$ be the i^{th} row of $\mathbf{A}$. Define $\varepsilon(x) = x - \lfloor x \rceil$, so that $\varepsilon(x) \in [-1/2, 1/2]$. We proceed by direct computation.

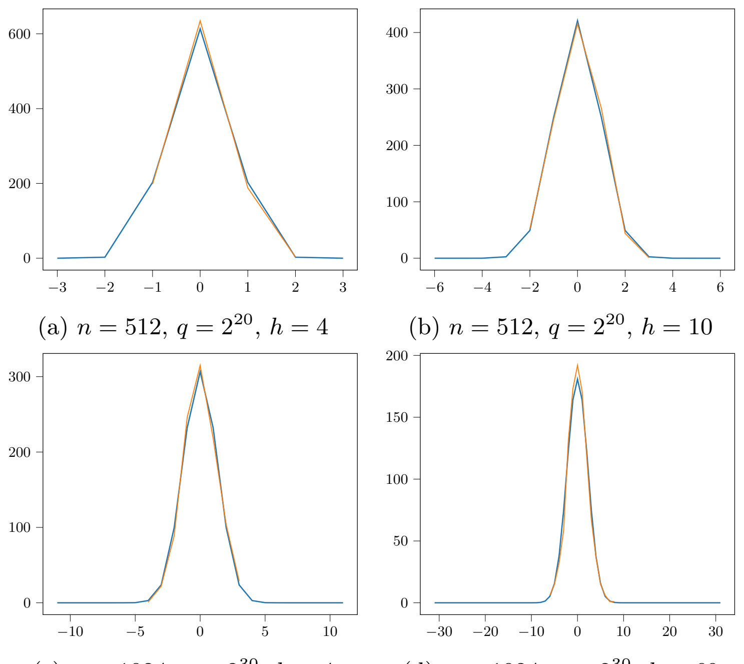
$$\begin{aligned}
\tilde{\mathbf{e}}_i &= \lfloor \mathbf{b}_i \rceil_p - \left\langle \lfloor \mathbf{A}_i \rceil_p, \mathbf{s} \right\rangle \bmod^{\pm} p \\
&= \left\lfloor \frac{p}{q}\mathbf{b}_i \right\rceil - \left\langle \left\lfloor \frac{p}{q}\mathbf{A}_i \right\rceil, \mathbf{s} \right\rangle \bmod^{\pm} p \\
&= \frac{p}{q}\mathbf{b}_i - \varepsilon\left(\frac{p}{q}\mathbf{b}_i\right) - \left\langle \frac{p}{q}\mathbf{A}_i, \mathbf{s} \right\rangle + \left\langle \varepsilon\left(\frac{p}{q}\mathbf{A}_i\right), \mathbf{s} \right\rangle \bmod^{\pm} p \\
&= \frac{p}{q}\mathbf{e}_i - \varepsilon\left(\frac{p}{q}\mathbf{b}_i\right) + \left\langle \varepsilon\left(\frac{p}{q}\mathbf{A}_i\right), \mathbf{s} \right\rangle \bmod^{\pm} p. \qquad (\text{by } \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q)
\end{aligned}$$

Heuristically, we assume that $\varepsilon((p/q) \cdot \mathbf{A}_i) \sim_{\text{i.i.d.}} U(-1/2, 1/2)$ and similarly that $\varepsilon((p/q) \cdot \mathbf{b}_i) \sim U(-1/2, 1/2)$, distributed independently from the other terms. Since $\mathbf{s}$ has exactly h non-zero coefficients, all in $\{-1, 1\}$, $\langle \varepsilon((p/q) \cdot \mathbf{A}_i), \mathbf{s} \rangle$ is a sum of h continuous random variables $\sim U(-1/2, 1/2)$. It follows that, heuristically,

$$\begin{aligned}
\tilde{\mathbf{e}}_i &\sim \frac{p}{q}\mathbf{e}_i + \sum_{j=1}^{h+1} U\left(-\frac{1}{2}, \frac{1}{2}\right) \bmod^{\pm} p \\
&\approx \mathsf{CIH}_{h+1} \bmod^{\pm} p && (\text{by } p\sigma_e \ll q) \\
&= \mathsf{CIH}_{h+1} && (\text{by } h + 1 < p)
\end{aligned}$$

Of course, this is not technically possible, since $\tilde{\mathbf{e}}_i$ is by definition an integer, while CIH_{h+1} follows a continuous distribution. Therefore, we conjecture that in practice $\tilde{\mathbf{e}}_i \sim \mathsf{DCIH}_{h+1}$. □

In Fig. 2 experimental evidence is given for the accuracy of Heuristic 5.1. In particular, for various $(n, p, q, \chi_e, \chi_s)$ the actual error in the implied LWE instance after rounding as in Section 5.2 is compared to the heuristically assumed DCIH distribution.



(a) $n = 512$, $q = 2^{20}$, $h = 4$

(b) $n = 512$, $q = 2^{20}$, $h = 10$

(c) $n = 1024$, $q = 2^{30}$, $h = 4$

(d) $n = 1024$, $q = 2^{30}$, $h = 69$

Fig. 2: Experiments verifying Heuristic 5.1. All LWE instances have ternary secrets with h non-zero coefficients, $m = 1024$, $\sigma_e = 3.2$. Experiments (2a) and (2b) round down to $p = 1024$, (2c) rounds down to $p = 512$, and (2d) down to $p = 2^{20}$. The lines are histograms for the DCIH_{h+1} distribution (blue) and measured $\tilde{\mathbf{e}}_i$ values (orange).

E Implementing the primal attack against sparse secrets, or how not to misrepresent baseline costs

In [WSM$^+$25], the authors benchmark a series of recently proposed attacks on sparse secret LWE, announcing that C+C clearly outperforms the state of the art primal attack. To do so, the authors identify various extremely sparse secret instances of LWE, estimate the hardness of the primal attack (for $\mathbf{x} = (c\mathbf{s}, \mathbf{e})$) with a modified version of the lattice-estimator that considers attacks in the $20 \leq \beta < 40$ regime [WSM$^+$25, Tab. 10], and instantiate the attack on high dimensional lattice embeddings. They observe that this strategy does not lead to termination within 1300 core-hours on their machines, reporting it as unsuccessful. They then proceed to evaluate C+C and machine learning-based attacks with significantly higher core-hour counts and with low success probabilities, reporting them as successful [WSM$^+$25, Tab. 9]. From these observations, they conclude that these new techniques outperform the baseline primal attack.

As described in Section 5, sparse secret LWE usually benefits from the secret dimension vs. success probability trade-off of Drop + Solve, as it can meaningfully reduce the dimension of the lattice problem. However, [WSM$^+$25] does not attempt to estimate using the {LWE, lattice}-estimator, nor measure (with experiments) this trade-off. Their cost predictions are also in the inaccurate $\beta < 40$ regime, which further complicates a comparison with state-of-the-art implementations of the primal attack. Finally, they perform comparisons between the cost generated by the estimator using asymptotic cost models for the number of nodes visited during lattice enumeration with extreme pruning, and core-hours in the $\beta \approx 50$ regime, claiming these predictions do not match their experiments [WSM$^+$25, App. B]. While these units of measure are not entirely incomparable, much more contextual information is required, such as the internals of the enumeration implementation, and the cost of running these internals on their specific hardware. The authors' false equivalence between ring operations ('rop') and CPU clock cycles instead contributes negatively to their predictions' accuracy. The resulting claimed costs therefore do not represent a meaningful baseline for the wall-time required to successfully mount the primal attack on sparse secret LWE.

F A detailed view of the Drop + Solve attack

F.1 Description

Let $I \subset [n]$ be a set of indices of $\mathbf{s}$ to keep in $\mathbf{x}$ and $J = [n] \setminus I$. Write $\mathbf{s}_I = (s_i)_{i \in I}$ and similarly $\mathbf{s}_J$. Without loss of generality, suppose $J = [k]$ and $I = k + [n-k]$ and write $\mathbf{A} = (\mathbf{A}_J, \mathbf{A}_I)$ with $\mathbf{A}_J \in \mathbb{Z}_q^{m \times k}$ and $\mathbf{A}_I \in \mathbb{Z}_q^{m \times (n-k)}$. To encode only $\mathbf{s}_I$ in $\mathbf{x}$, guess $\widetilde{\mathbf{s}}_J$ for the value of $\mathbf{s}_J$ and construct the tuple $(\mathbf{A}_I, \mathbf{b}')$, with $\mathbf{b}' = \mathbf{b} - \mathbf{A}_J \widetilde{\mathbf{s}}_J$. If the guess is correct, meaning $\mathbf{s}_J = \widetilde{\mathbf{s}}_J$, then $(\mathbf{A}_I, \mathbf{b}')$ is a new LWE instance with m samples, secret dimension $|I| < n$ and secret $\mathbf{s}_I$. Indeed, $\mathbf{b}' = \mathbf{b} - \mathbf{A}_J \mathbf{s}_J = \mathbf{A}_I \mathbf{s}_I + \mathbf{e} \bmod q$. Therefore $\mathbf{s}_I$ can be recovered using the primal attack against this lower dimensional LWE instance. If the guess is incorrect, and $\widetilde{\mathbf{s}}_J = \mathbf{s}_J + \mathbf{d}$ for some unknown $\mathbf{d} \neq \mathbf{0}$, then $\mathbf{b}' = \mathbf{A}_I \mathbf{s}_I + \mathbf{e} - \mathbf{A}_J \mathbf{d}$, and the $\mathbf{A}_J \mathbf{d}$ term essentially 'randomises' the LWE instance such that lattice reduction cannot recover $\mathbf{x}$. Since $\mathbf{s}_J$ needs to be guessed correctly this attack has low success probability, and needs to be repeated for different guesses $\widetilde{\mathbf{s}}_J$, presenting a secret dimension vs. success probability trade-off. When $\mathbf{s}$ is very sparse, this approach generally results in a significant speedup, as it is possible to guess J such that $\mathbf{s}_J = (0, \ldots, 0)$ with high probability for a relatively large k.

F.2 Choosing attack parameters

In [WSM+25, Table 12], we see experiments reported for the following ring-LWE parameters

$$n = 1024,\ \log_2 q = 26,\ \chi_e = N(0, \sigma^2) \text{ with } \sigma = 3.19,\ m = 4n,$$

and χ_s uniform in $\{-1, 0, 1\}^n$ subject to having only $7 \leq h \leq 12$ non zero coordinates. Following the Drop + Solve strategy we sample index sets J of cardinality k, to be determined, guessing that $\mathbf{s}_J = (0, \ldots, 0)$. By sampling J at random, to avoid having to keep a potentially very large array listing already made guesses, each attempt has success probability $p_{\text{guess}} = \binom{n-h}{k} / \binom{n}{k}$. Supposing lattice reduction on a lattice embedding obtained from a valid guess for J recovers $\mathbf{x}$ with probability $p_{\text{red}} \in (0, 1]$ independent of J, we expect to have to make approximately $(p_{\text{guess}} \cdot p_{\text{red}})^{-1}$ guesses before solving LWE, with a tail bound of at most $\log(1-p)/\log(1 - p_{\text{guess}} p_{\text{red}})$ attempts to solve LWE with probability p. This analysis is only a starting point, and proposes many questions. How do we find a 'good' value for k? How do we estimate p_{red}? What does the resulting embedding look like? How do we efficiently reduce it?

Finding k. Supposing the attacker has a machine with many cores, the simplest way of implementing the above attack is to use every core to make independent guesses of J then perform the relevant lattice reduction. How effective this trade-off is on a specific machine will depend on the wall-time of performing lattice reduction with probability p_{red}, which mainly depends on the wall-time of solving the Shortest Vector Problem (SVP) on that machine. We do not attempt to resolve the problem of accurate wall-time predictions of the runtime of small block size lattice reduction here. However, we note that realistically as part of small experiments, we can afford at most running a few thousand guesses per LWE instance.

Estimating p_{red}. The success probability of progressive-BKZ [AWHT16] has been studied in [DDGR20] and its analysis has been adapted to the case of BKZ in [PV21]. In these works, estimators are put forward that simulate BKZ reduction and extrapolate estimates for p_{red} whenever $\beta \geq 60$, the accuracy of which is then experimentally verified. To obtain a direct comparison with [WSM+25], we only consider attacks reaching $\beta \leq 45$, for which these predictions are inaccurate. From inspection, we observed that in our initial experiments $p_{\text{red}} \approx 0.7$, and used this value when choosing attack parameters.

Building an embedding. Now suppose that we choose k and obtain estimates for p_{red}. After making a guess for J we obtain a new LWE instance $(\mathbf{A}', \mathbf{b}')$ with an $(n-k)$-dimensional secret $\mathbf{s}'$ with h non zero entries, and an error distribution of standard deviation $\sigma_e = 3.19$. Likely, for such sparse instances, the resulting distribution $\chi_{s'}$ of $\mathbf{s}'$ is still significantly narrower than χ_e. Therefore, we choose a scaling factor $c > 1$ to even the contribution of $\mathbf{s}'$ and $\mathbf{e}$ to $\mathbf{x}$. To map the resulting lattice into an integer lattice suitable for reduction using the FPLLL library,[17] we choose a rational approximation $\bar{c} \in \mathbb{Q}$ of c with approximately 1% error, and multiply the embedding lattice by the denominator of $\bar{c}$.

Performing lattice reduction. Finally, we are left with a lattice to reduce. Most likely, our guess for J is incorrect, meaning that we perform lattice reduction until termination. It is therefore crucial to reduce the practical cost of lattice reduction. While asymptotically it is calls to the SVP algorithm that dominate the cost of lattice reduction, when running practical experiments it is large dimensions or moduli that have a significant impact due to the resulting requirement for high precision floating point arithmetic. Flatter was introduced in [RH23], a fast lattice reduction algorithm that outputs bases reduced in a similar fashion to LLL. Flatter is able to handle high dimensions and moderately large moduli within an acceptable runtime. While Flatter is not integrated within FPLLL's BKZ, the most costly LLL call in primal attacks is the one used to preprocess the embedding basis before running BKZ. Hence, one may call Flatter independently from FPLLL then pass the output to the latter library.

F.3 Drop + Solve algorithm

In Algorithm 5 we give pseudocode for the Drop + Solve algorithm implemented for the experiments of Section 5.4.

G Experimental setup and results reporting

G.1 Setting up Drop + Solve

In the case of Drop + Solve, our approach is to:

1. Generate LWE instances modulo q.
2. If $q \geq 2^{30}$ then round the instances modulo $p = 2^{10}$ or $p = 2^{15}$, following Section 5.2, noting the new induced error variance $\frac{h+1}{12}$.

[17] https://github.com/fplll/fplll

Algorithm 5: Implemented Drop + Solve

```
Input: LWE instance $\mathbf{A}, \mathbf{b}$ over $\mathbb{Z}_q$, $k \in \mathbb{N}$, $p < q$, sparse secret distribution
       $\chi_s$, error distribution $\chi_e$ with entrywise standard deviation $\sigma$,
       maximum blocksize $\beta$
Output: The secret $\mathbf{s}$ of the LWE instance
1  Procedure KannanEmbed($\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{b} \in \mathbb{Z}_q^m$):
2      Choose an appropriate scaling $c' = \frac{c_1}{c_2} \in \mathbb{Q}$
3      Build $\mathbf{B}$, a basis for Eq. (2)
           $$\mathbf{B} = \begin{bmatrix} c_1\mathbf{I}_n & \\ -c_2\mathbf{A} & c_2 q \mathbf{I}_m \end{bmatrix}$$
4      Embed target $\mathbf{t} = (0^n, \mathbf{b})^T$ to form $\mathbf{B}'$
5
           $$\mathbf{B}' = \begin{bmatrix} \mathbf{B} & \mathbf{t} \\ \mathbf{0} & \lceil \sigma \rfloor \end{bmatrix}$$
6      $\mathbf{B}'_{\mathtt{BKZ}} \leftarrow$ progressive−BKZ($\mathbf{B}'$) up to blocksize $\beta$
7      Return the shortest vector from $\mathbf{B}'_{\mathtt{BKZ}}$
8  repeat
9      Choose uniform $J \subset [n]$ such that $|J| = k$
       /* Apply $q$−to−$p$ rounding the columns of A indexed by $[n] \setminus J$ */
10     $\mathbf{A}_p \leftarrow \lfloor \mathbf{A}_{[n]\setminus J} \rceil_p$
11     $\mathbf{b}_p \leftarrow \lfloor \mathbf{b} \rceil_p$
12     Let $\chi'_s$ be $\chi_s$ with entries indexed by $J$ removed
13     $\mathbf{x} \leftarrow$ KannanEmbed($\mathbf{A}_p, \mathbf{b}_p$)
14     if $\mathbf{x}$ is correct with respect to $c'$, $\chi'_s$ and $\chi_e$ then
15         Let $\mathbf{x} = (c_1\mathbf{s}', c_2\mathbf{e})$
16         Return $\mathbf{s}$ formed of $\mathbf{s}'$ with zero entries added according to $J$
17 until Return $\mathbf{s}$
```

3. Estimating $p_{\text{red}} \approx 0.7$, choose k entries to guess such that the plausible number of guesses $(p_{\text{guess}} \cdot p_{\text{red}})^{-1} \leq 1000$.
4. For each such k, use the [ADPS16] methodology for estimating the required block size β to break the LWE instance.
5. For each generated instance, perform progressive-BKZ reduction up to the block size β.

For the sake of comparing Drop + Solve with non-guessing primal attacks, for some parameter sets we also run attacks with $k = 0$. Throughout, the instances we choose seem to be solvable in the small block size regime, $\beta \leq 45$. As we will see next, this is a similar regime to that used in C+C, therefore we do not push for instances that require larger block sizes to be solved.

G.2 Setting up C+C

Due to the lack of a public runtime analysis or comprehensive theoretical analysis, we could not find clear instructions for choosing optimal C+C parameters. The C+C codebase exposes a large amount of attack parameters to be chosen by the user. Most have default values, though not all are mentioned in either [WSM+25] or [NMW+24]. The [WSM+25] code release does contain some examples for calling the attack on toy parameters, but not a script for systematically reproducing the benchmarks reported in the paper.

We attempted to choose fair attack parameters, balancing those reported in [NMW+24], [WSM+25], and in the code release. Our choices are as follows:

1. Preprocessing BKZ block sizes: [NMW+24, Tab. 4] proposes 35 and 40 for dimension 256, and 18 and 22 for dimensions at least 512. The code release README gives 32 and 38 for dimension 128, while the default values in `preprocessing.py` are 30 and 40. Since 30 and 40 are similar to those that our Drop + Solve instances require, and are default values, we use these.
2. LLL penalty ω: this is our scaling factor c. [NMW+24] uses $\omega \in \{4, 10\}$, while [WSM+25] mentions $\omega \in \{1, 4, 10\}$, with $\omega = 10$ being used for 'FHE' parameters (large q, sparse ternary secrets). Due to the nature of our instances, we also set $\omega = 10$.
3. Lattice reduction thresholds ρ: in [NMW+24,WSM+25] the description of C+C appears contradictory regarding ρ. In [NMW+24] it is suggsted that ρ is estimated from the output of the preprocessing phase, yet the [WSM+25] code release requires three non decreasing values of ρ as input to the preprocessing. A variety of options appear; $\rho \in \{0.86, 0.84, 0.70\}$ for FHE instances in [WSM+25], $\rho \in [0.413, 0.769]$ in [NMW+24], $\rho \approx 0.783$ in the README file of the code release (for $n = 80$), and $\rho \in \{0.4, 0.41, 0.5\}$ as default values in `preprocessing.py`. The latter default values are much lower than most other settings, and do not appear to be used. We instead strike a balance between the other values and choose the example value of $\rho \approx 0.783$ from the README file.

4. LWE samples m: C+C is described as working well when given in input a matrix $\mathbf{A} \in \mathbb{Z}_q^{4n \times n}$, allowing it to subsample many $\mathbf{A}_i \in \mathbb{Z}_q^{m \times n}$ for $m = \lceil 7n/8 \rfloor$ (see Algorithm 3). However, we also explore a setting where C+C is only provided $2n$ samples (as is common in LWE based PKE), and it is asked to sample many $\mathbf{A}_i$ with m as used in Drop + Solve.
5. Amount of preprocessing: in [NMW$^+$24, Eq. 11] the authors provide an estimate for the number of LWE samples that should be output during preprocessing to lead to a successful C+C attack. In [NMW$^+$24, Tab. 5] approximately $11,000$ is suggested as a sweet spot. However, we were not able to request a maximum number of samples in `preprocessing.py` via its interface. Instead, we opt to limit the amount of preprocessing by wall time, as we explain below.

In order to perform a fair comparison, we use the same LWE input distributions for both attacks. In particular, this means that when $q \geq 2^{30}$, LWE samples are rounded down modulo p also in the case of C+C.

G.3 Measurements.

Measuring the performance of Drop + Solve is relatively easy. Given our estimate $p_{\text{red}} \cdot p_{\text{guess}}$ for the probability of successfully breaking LWE with a specific guess, by modelling different guesses as independent we can obtain a tail bound for the number N of attempts to break LWE with 99% accuracy. We then generate 10 LWE instances, and for each we let Drop + Solve perform up to N guesses. If LWE is not solved, we count that call to Drop + Solve as a failure. We then measure the proportion of successful runs of Drop + Solve, and the average runtime required for running those successful instances. It should be noted that our implementation uses 94 out of 96 available cores in parallel, having each perform a different guess. In Table 2, we report average wall times.

Since we could not find a simple way to limit the amount of preprocessing performed by the C+C implementation, we could not just simply let the attack run, count successes and failures, and measure average runtimes. Instead, we opt to compare success probabilities when running C+C for a comparable amount of time to Drop + Solve. Specifically, if Drop + Solve's average wall time was μ_t and its sample variance was σ_t^2, we define a 'timeout' $T = \lceil \max(1.1 \cdot \mu_t, \mu_t + \sigma_t) \rfloor$. We then let preprocessing run for at most T time using 94 out of 96 cores in parallel, and then let secret guessing also run for at most T time. We observe that most of the time is spent in preprocessing, which performs massively parallel lattice reduction, similar to what Drop + Solve does. Therefore, overall C+C is allowed to run for a little longer than Drop + Solve on the same LWE instances, and using approximately the same resources. We then compare success probabilities.

G.4 Time reporting for Guess + Verify and Cool + Cruel

In this subsection, we explain the performed computations of the reported numbers in the Guess + Verify and Cool + Cruel comparison of Table 2. First, we explain why the time of the Cool + Cruel attack has error bars, and then we explain how the number of core-hours and number of GPU-hours is computed in the Guess + Verify attack.

The Cool + Cruel timings are taken from [WSM+25, Table 9,11,12], which contains the wall time in hours, the number of used CPUs and the number of used GPUs. These timings are reported with 1, 2 or 3 significant digits, so the exact minimum wall time could be in a wide range. For example, the first column [WSM+25, Table 9] reports that Cool + Cruel ran for 0.1 hours on 256 GPUs. Since the wall time can be any number in the range $[0.05, 0.15]$, the total GPU-hours can be anywhere in the range 12.8h–38.4h. The reported mean is obtained by multiplying wall time with the number of GPUs. On the other hand, the uncertainty in the GPU-hours, is then computed by multiplying the uncertainty in the reported number with the number of GPUs, and then rounding this up.

Table 3 reports the average wall time taken by the Guess + Verify attack to break the instances of Table 2. These experiments ran on the servers listed in Table 1. Yet, we used fewer computing cores/GPUs than available: Y used 92 CPU cores, Z used 8 CPU cores and 8 GPUs, and H used 40 CPU cores from one CPU and 2 GPUs.

Table 3: Average wall time, core-hours and GPU-hours for the Guess + Verify attack on instances of Table 2.

MLWE parameters					**Guess + Verify**				
χ_s	n	q	h	σ_e	succ.	time (s)	core-hours	GPU-hours	server
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{12}$	11	1	5 / 6	20 448	≤ 522.5	5.7	Y
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{12}$	12	1	2 / 2	124 901	$\leq 3\,191.9$	34.7	Y
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{28}$	20	1	4 / 5	10 158	≤ 22.6	22.6	Z
$\mathfrak{B}^2$	$2 \cdot 256$	$\approx 2^{28}$	21	1	5 / 5	12 547	≤ 27.9	27.9	Z
Ter	$1 \cdot 1024$	$\approx 2^{26}$	11	3.19	5 / 5	32 584	≤ 832.7	9.1	Y
Ter	$1 \cdot 1024$	$\approx 2^{29}$	9	3.19	5 / 5	16 841	≤ 187.1	9.4	H
Ter	$1 \cdot 1024$	$\approx 2^{29}$	10	3.19	5 / 5	76 073	≤ 845.3	42.3	H