LWE with Side Information: Attacks and Concrete Security Estimation[⋆]

Dana Dachman-Soled[1] and Léo Ducas[2] and Huijing Gong[1] and Mélissa Rossi[3,4,5,6]

[1] University of Maryland, College Park, USA
gong@cs.umd.edu
danadach@ece.umd.edu
[2] CWI, Amsterdam, The Netherlands
[3] ANSSI, Paris, France
[4] ENS Paris, CNRS, PSL University, Paris, France
[5] Thales, Gennevilliers, France
[6] INRIA, Paris, France
melissa.rossi@ens.fr

Abstract. We propose a framework for cryptanalysis of lattice-based schemes, when side information—in the form of "hints"— about the secret and/or error is available. Our framework generalizes the so-called primal lattice reduction attack, and allows the progressive integration of hints before running a final lattice reduction step. Our techniques for integrating hints include sparsifying the lattice, projecting onto and intersecting with hyperplanes, and/or altering the distribution of the secret vector. Our main contribution is to propose a toolbox and a methodology to integrate such hints into lattice reduction attacks and to predict the performance of those lattice attacks with side information.

While initially designed for side-channel information, our framework can also be used in other cases: exploiting decryption failures, or simply exploiting constraints imposed by certain schemes (LAC, Round5, NTRU), that were previously not known to (sligthly) benefit from lattice attacks.

We implement a Sage 9.0 toolkit to actually mount such attacks with hints when computationally feasible, and to predict their performances on larger instances. We provide several end-to-end application examples, such as an improvement of a single trace attack on Frodo by Bos et al (SAC 2018). Contrary to ad-hoc practical attacks exploiting side-channel leakage, our work is a generic way to estimate security loss even given very little side-channel information.

Keywords: LWE, NTRU, Lattice reduction, Cryptanalysis, Side-channels analysis, decryption failures.

1 Introduction

A large effort is currently underway to replace standardized public key cryptosystems, which are quantum-insecure, with newly developed "post-quantum" cryptosystems, conjectured to be secure against quantum attack. Lattice-based cryptography has been widely recognized as a foremost candidate for practical, post-quantum security and accordingly, a large effort has been made to develop and analyze lattice-based cryptosystems. The ongoing standardization process and anticipated deployment of lattice-based cryptography raises an important question: How resilient are lattices to side-channel attacks or other forms of side information? While there are numerous works addressing this question for specific cryptosystems (See [2,12,20,36,35,11] for

[⋆] The research of L. Ducas and M. Rossi was supported by the European Union's H2020 Programme under PROMETHEUS project (grant 780701). The research of M. Rossi was also supported by ANRT under the programs CIFRE N 2016/1583. It was also supported by the French Programme d'Investissement d'Avenir under national project RISQ P14158. The research of D. Dachman-Soled and H. Gong was supported in part by NSF grants #CNS-1933033, #CNS-1840893, #CNS-1453045 (CAREER), by a research partnership award from Cisco and by financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology.

$$\begin{tikzcd}\mathsf{LWE/BDD} \arrow[r, "\text{Kannan}"] & \mathsf{uSVP}_{\Lambda'} \arrow[r, "\text{Sec 3.4}"] & \text{Lattice reduction}\end{tikzcd}$$

Fig. 1. Primal attack without hints (prior art).

side channel attacks targeting lattice-based NIST candidates), these works use rather ad-hoc methods to reconstruct the secret key, requiring new techniques and algorithms to be developed for each setting. For example, the work of [11] uses brute-force methods for a portion of the attack, while [10] exploits linear regression techniques. Moreover, ad-hoc methods do not allow (1) to take advantage of decades worth of research and (2) optimization of standard lattice attacks. Second, most of the side-channel attacks from prior work consider substantial amounts of information leakage and show that it leads to feasible recovery of the entire key, whereas one may be interested in more precise tradeoffs in terms of information leakage versus concrete security of the scheme. The above motivates the focus of this work: Can one integrate side information into a standard lattice attack, and if so, by how much does the information reduce the cost of this attack? Given that side-channel resistance is the next step toward the technological readiness of lattice-based cryptography, and that we expect numerous works in this growing area, we believe that a general framework and a prediction software are in order.

Contributions. First, we propose a framework that generalizes the so-called primal lattice reduction attack, and allows the progressive integration of “hints” (i.e. side information that takes one of several forms) before running the final lattice reduction step. This contribution is summarized in Figures 1 and 2 and developed in Section 3.

Second, we implement a Sage 9.0 toolkit to actually mount such attacks with hints when computationally feasible, and to predict their performance on larger instances. Our predictions are validated by extensive experiments. Our tool and these experiments are described in Section 5. Our toolkit is open-source, available at: `https://github.com/lducas/leaky-LWE-Estimator`.

Third, we demonstrate the usefulness of our framework and tool via three example applications. Our main example (Section 6.1) revisits the side channel information obtained from the first side-channel attack of [11] against Frodo. In that article, it was concluded that a divide-and-conquer side-channel template attack would not lead to a meaningful attack using standard combinatorial search for reconstruction of the secret. Our technique allows to integrate this side-channel information into lattice attacks, and to predict the exact security drop. For example, the CCS2 parameter set very conservatively aims for 128-bits of post-quantum security (or 448 “bikz” as defined in Section 3.4); but after the leakage of [11] we predict that its security drops to 110 “bikz”, i.e. that it can be broken with BKZ-110, a computation that should take about a few thousand core-hours with the latest lattice reduction library [3].

Interestingly, we note that our framework is not only useful in the side-channel scenario; we are for example also able to model decryption failures as hints fitting our framework. This allows us to reproduce some predictions from [16]. This is discussed in Section 6.2.

Perhaps more surprisingly, we also find a novel improvement to attack a few schemes (LAC [28], Round5 [19], NTRU [39]) without any side-channel or oracle queries. Indeed, such schemes use ternary distribution for secrets, with a prescribed numbers of 1 and -1: this hint fits our framework, and lead to a (very) minor improvement, discussed in Section 6.3.

Lastly, our framework also encompasses and streamlines existing tweaks of the primal attack: the choice of ignoring certain LWE equations to optimize the volume-dimension trade-off, as well as the re-centering [33] and isotropization [22,14] accounting for potential a-priori distortions of the secret. It also implicitly solves the question of the optimal choice of the coefficient for Kannan

$$\begin{tikzcd}
\mathsf{LWE/BDD} \arrow[r, "\text{Sec 3.2}"] & \mathsf{DBDD}_{\Lambda_0,\mathbf{\Sigma}_0,\boldsymbol{\mu}_0} \arrow[d, "\text{Sec 4}"', "\textbf{Hint}"] & & \\
& \mathsf{DBDD}_{\Lambda_1,\mathbf{\Sigma}_1,\boldsymbol{\mu}_1} & & \\
& \vdots \arrow[d, "\text{Sec 4}"', "\textbf{Hint}"] & & \\
& \mathsf{DBDD}_{\Lambda_h,\mathbf{\Sigma}_h,\boldsymbol{\mu}_h} \arrow[r, "\text{Sec 3.3}"] & \mathsf{uSVP}_{\Lambda'} \arrow[r, "\text{Sec 3.4}"] & \text{Lattice reduction}
\end{tikzcd}$$

Fig. 2. The primal attack with hints (our work).

Bounded Distance Decoding problem (BDD) to unique Shortest Vector Problem (uSVP) [24] (See Remark 23).

As a side contribution, we also propose in Section 3.4a refined method to estimate the required blocksize to solve an LWE/BDD/uSVP instance. This refinement was motivated by the high inaccuracy of the standard method from the literature [4,5] in experimentally reachable blocksizes, which was making the validation of our contribution difficult. While experimentally much more accurate, this new methodology certainly deserves further scrutiny.

Technical overview. Our work is based on a generalization of the Bounded Distance Decoding problem (BDD) to a Distorted version (DBDD), which allows to account for the potentially non-spherical covariance of the secret vector to be found.

Each hint will affect the lattice itself, the mean and/or the covariance parameter of the DBDD instance, making the problem easier (see Figure 2). At last, we make the distribution spherical again by applying a well-chosen linear transformation, reverting to a spherical BDD instance before running the attack. Thanks to the hints, this new instance will be easier than the initial one. Let us assume that $\mathbf{v}$, l, k and σ are parameters known by the attacker. Our framework can handle four types of hints on the secret $\mathbf{s}$ or on the lattice Λ.

– Perfect hints: $\langle \mathbf{s}, \ \mathbf{v} \rangle = l$ *intersect the lattice with an hyperplane.*
– Modular hints : $\langle \mathbf{s}, \ \mathbf{v} \rangle = l \bmod k$ *sparsify the lattice.*
– Approximate hints : $\langle \mathbf{s}, \ \mathbf{v} \rangle = l + \epsilon_\sigma$ *decrease the covariance of the secret.*
– Short vector hints : $\mathbf{v} \in \Lambda$ *project orthogonally to* $\mathbf{v}$.

While the first three hints are clear wins for the performance of lattice attacks, the last one is a trade-off between the dimension and the volume of the lattice. This last type of hint is in fact meant to generalize the standard trick consisting of 'ignoring' certain LWE equations; ignoring such an equation can be interpreted geometrically as such a projection orthogonally to a so-called q-vector.

All the transformations of the lattice above can be computed in polynomial time. However, computing with general distribution in large dimension is not possible; we restrict our study to the case of Gaussian distributions of arbitrary covariance, for which such computations are also poly-time.

Some of these transformations remain quite expensive, in particular because they involve rational numbers with very large denominators, and it remains rather impractical to run them on cryptographic-grade instances. Fortunately, up to a necessary hypothesis of primitivity of the vector $\mathbf{v}$ (with respect to either Λ or its dual depending on the type of hint), we can also predict the effect of each hint on the lattice parameters, and therefore run faster predictions of the attack cost.

From Leaks to Hints. At first, it may not be so clear that the types of hints above are so useful in realistic applications, in particular since they need to be linear on the secret. Of course our framework can handle rather trivial hints such as the perfect leak of a secret coefficient $\mathbf{s}_i = l$. Slightly less trivial is the case where the only the low-order bits leaks, a hint of the form $\mathbf{s}_i = l \bmod 2$.

We note that most of the computations done during an LWE decryption are linear: leaking any intermediate register during a matrix vector product leads to a hint of the same form (possibly mod$q$). Similarly, the leak of a NTT coefficient of a secret in a Ring/Module variant can also be viewed as such.

Admittedly, such ideal leaks of a full register are not the typical scenario and leaks are typically not linear on the content of the register. However, such non-linearities can be handled by approximate hints. For instance, let $\mathbf{s}_0$ be a secret coefficient (represented by a signed 16-bits integer), whose a priori distribution is supported by $\{-5, \dots, 5\}$. Consider the case where we learn the Hamming weight of $\mathbf{s}_0$, say $H(\mathbf{s}_0) = 2$. Then, we can narrow down the possibilities to $\mathbf{s}_0 \in \{3, 5\}$. This leads to two hints:

– a modular hint: $\mathbf{s}_0 = 1 \bmod 2$,
– an approximate hint: $\mathbf{s}_0 = 4 + \epsilon_1$, where ϵ_1 has variance 1.

While closer to a realistic scenario, the above example remains rather simplified. A detailed example of how realistic leaks can be integrated as hint will be given in Section 6.1, based on the leakage data from [11].

Acknowlegments.

The authors would like to thank Marco Martinoli and his co-authors [11] for sharing their source code. We express our gratitude to Jan-Pieter D'anvers for sharing precious insights and intuitions, guiding toward the proper formalization of decryption failures as approximate hints. We are also grateful to Martin Albrecht, Henri Gilbert, Ange Martinelli and Thomas Prest for valuable feedback on a preliminary version of this work.

2 Preliminaries

2.1 Linear Algebra

We use bold lower case letters to denote vectors, and bold upper case letters to denote matrices. We use row notations for vectors, and start indexing from 0. Let $\mathbf{I}_d$ denote the d-dimensional identity matrix. Let $\langle \cdot, \cdot \rangle$ denote the inner product of two vectors of the same size. Let us introduce the row span of a matrix (denoted Span$(\cdot)$) as the subspace generated by all $\mathbb{R}$-linear combinations of the rows of its input.

Definition 1 (Positive Semidefinite). *A $n \times n$ symmetric real matrix $\mathbf{M}$ is positive semidefinite if scalar $\mathbf{x}\mathbf{M}\mathbf{x}^T \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$; if so we write $\mathbf{M} \geq 0$. Given two $n \times n$ real matrix $\mathbf{A}$ and $\mathbf{B}$, we note $\mathbf{A} \geq \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is positive semidefinite.*

Definition 2. *A matrix $\mathbf{M}$ is a square root of $\mathbf{\Sigma}$, denoted $\sqrt{\mathbf{\Sigma}}$, if*

$$\mathbf{M}^T \cdot \mathbf{M} = \mathbf{\Sigma},$$

Our techniques involve keeping track of the covariance matrix $\mathbf{\Sigma}$ of the secret and error vectors as hints are progressively integrated. The covariance matrix may become singular during this process and will not have an inverse. Therefore, in the following we introduce some degenerate notions for the inverse and the determinant of a square matrix. Essentially, we restrict these notions to the row span (denoted Span()) of their input. For $\mathbf{X} \in \mathbb{R}^{d \times k}$ (with any $d, k \in \mathbb{N}$), we will denote $\mathbf{\Pi_X}$ the orthogonal projection matrix onto $\mathrm{Span}(\mathbf{X})$. More formally, let $\mathbf{Y}$ be a maximal set of indepedant row-vectors of $\mathbf{X}$; the orthogonal projection matrix is given by $\mathbf{\Pi_X} = \mathbf{Y}^T \cdot (\mathbf{Y} \cdot \mathbf{Y}^T)^{-1} \cdot \mathbf{Y}$. Its complement (the projection orthogonally to $\mathrm{Span}(\mathbf{X})$) is denoted $\mathbf{\Pi}_\mathbf{X}^\perp := \mathbf{I}_d - \mathbf{\Pi_X}$. We naturally extend the notation $\mathbf{\Pi}_F$ and $\mathbf{\Pi}_F^\perp$ to subspaces $F \subset \mathbb{R}^d$. By definition, the projection matrices satisfy $\mathbf{\Pi}_F^2 = \mathbf{\Pi}_F$, $\mathbf{\Pi}_F^T = \mathbf{\Pi}_F$ and $\mathbf{\Pi}_F \cdot \mathbf{\Pi}_F^\perp = \mathbf{\Pi}_F^\perp \cdot \mathbf{\Pi}_F = \mathbf{0}$.

Definition 3 (Restricted inverse and determinant). *Let $\mathbf{\Sigma}$ be a symmetric matrix. We define a* restricted inverse *denoted $\mathbf{\Sigma}^\sim$ as*

$$\mathbf{\Sigma}^\sim := (\mathbf{\Sigma} + \mathbf{\Pi}_\mathbf{\Sigma}^\perp)^{-1} - \mathbf{\Pi}_\mathbf{\Sigma}^\perp.$$

It satisfies $\mathrm{Span}(\mathbf{\Sigma}^\sim) = \mathrm{Span}(\mathbf{\Sigma})$ *and* $\mathbf{\Sigma} \cdot \mathbf{\Sigma}^\sim = \mathbf{\Pi}_\mathbf{\Sigma}$*.*

We also denote $\mathrm{rdet}(\mathbf{\Sigma})$ *as the* restricted determinant *defined as follows.*

$$\mathrm{rdet}(\mathbf{\Sigma}) := \det(\mathbf{\Sigma} + \mathbf{\Pi}_\mathbf{\Sigma}^\perp).$$

The idea behind Definition 3 is to provide an (artificial) invertibility property to the input $\mathbf{\Sigma}$ by adding the missing othogonal part and to remove it afterwards. For example, if $\mathbf{\Sigma} = \begin{bmatrix} \mathbf{A} & 0 \\ 0 & 0 \end{bmatrix}$ where $\mathbf{A}$ is invertible,

$$\mathbf{\Sigma}^\sim = \left(\begin{bmatrix} \mathbf{A} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right)^{-1} - \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & 0 \\ 0 & 0 \end{bmatrix} \text{ and } \mathrm{rdet}\,\mathbf{\Sigma} = \det(\mathbf{A}).$$

2.2 Statistics

Before hints are integrated, we will assume that the secret and error vectors follow a multidimensional normal (Gaussian) distribution. Hints will typically correspond to learning a (noisy, modular or perfect) linear equation on the secret. We must then consider the altered distribution on the secret, conditioned on this information. Fortunately, this will also be a multidimensional normal distribution with an altered covariance and mean. In the following, we present the precise formulae for the covariance and mean of these conditional distributions.

Definition 4 (Multidimensional normal distribution). *Let $d \in \mathbb{Z}$, for $\boldsymbol{\mu} \in \mathbb{Z}^d$ and $\mathbf{\Sigma}$ being a symmetric matrix of dimension $d \times d$, we denote by $\mathcal{D}^d_{\mathbf{\Sigma},\boldsymbol{\mu}}$ the multidimensional normal distribution supported by $\boldsymbol{\mu} + \mathrm{Span}(\mathbf{\Sigma})$ by the following*

$$\mathbf{x} \mapsto \frac{1}{\sqrt{(2\pi)^{\mathrm{rank}(\mathbf{\Sigma})} \cdot \mathrm{rdet}(\mathbf{\Sigma})}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}) \cdot \mathbf{\Sigma}^\sim \cdot (\mathbf{x} - \boldsymbol{\mu})^T \right).$$

The following states how a normal distribution is altered under linear transformation.

Lemma 5. *Suppose $\mathbf{X}$ has a $\mathcal{D}^d_{\mathbf{\Sigma},\boldsymbol{\mu}}$ distribution. Let $\mathbf{A}$ be a $n \times d$ matrix. Then $\mathbf{X}\mathbf{A}^T$ has a $\mathcal{D}^n_{\mathbf{A}\mathbf{\Sigma}\mathbf{A}^T,\boldsymbol{\mu}\mathbf{A}^T}$ distribution.*

Lemma 6 shows the altered distribution of a normal random variable conditioned on its noisy linear transformation value, following from [27, Equations (6) and (7)].

Lemma 6 (Conditional distribution $\mathbf{X}|\mathbf{X}\mathbf{A}^T + \mathbf{b}$ from [27]). *Suppose that $\mathbf{X} \in \mathbb{Z}^d$ has a $\mathcal{D}^d_{\boldsymbol{\Sigma},\boldsymbol{\mu}}$ distribution, and $\mathbf{b} \in \mathbb{Z}^d$ has a $\mathcal{D}^d_{\boldsymbol{\Sigma}_{\mathbf{b}},\mathbf{0}}$ distribution. Let us fix $\mathbf{A}$ as a $n \times d$ matrix and $\mathbf{y} \in \mathbb{Z}^n$. The conditional distribution of $\mathbf{X}\Big| \left(\mathbf{X}\mathbf{A}^T + \mathbf{b} = \mathbf{y}\right)$ is $\mathcal{D}^d_{\boldsymbol{\Sigma}',\boldsymbol{\mu}'}$, where*

$$\begin{aligned}\boldsymbol{\mu}' &= \boldsymbol{\mu} + (\mathbf{y} - \boldsymbol{\mu}\mathbf{A}^T)(\mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T + \boldsymbol{\Sigma}_{\mathbf{b}})^{-1}\mathbf{A}\boldsymbol{\Sigma}\\ \boldsymbol{\Sigma}' &= \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\mathbf{A}^T(\mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T + \boldsymbol{\Sigma}_{\mathbf{b}})^{-1}\mathbf{A}\boldsymbol{\Sigma}.\end{aligned}$$

Corollary 7 (Conditional distribution $\mathbf{X}|\langle\mathbf{X},\mathbf{v}\rangle + e$). *Suppose that $\mathbf{X} \in \mathbb{Z}^d$ has a $\mathcal{D}^d_{\boldsymbol{\Sigma},\boldsymbol{\mu}}$ distribution and e has a $\mathcal{D}^1_{\sigma_e^2,0}$ distribution. Let us fix $\mathbf{v} \in \mathbb{R}^d$ as a nonzero vector and $y \in \mathbb{Z}$. We define the following scalars:*

$$Y = \langle\mathbf{X},\mathbf{v}\rangle + e,\ \mu_2 = \langle\mathbf{v},\boldsymbol{\mu}\rangle \text{ and } \sigma_2 = \mathbf{v}\boldsymbol{\Sigma}\mathbf{v}^T + \sigma_e^2$$

If $\sigma_2 \neq 0$, the conditional distribution of $\mathbf{X}\Big|(Y = y)$ is $\mathcal{D}^d_{\boldsymbol{\Sigma}',\boldsymbol{\mu}'}$, where

$$\begin{aligned}\boldsymbol{\mu}' &= \boldsymbol{\mu} + \frac{(y - \mu_2)}{\sigma_2}\mathbf{v}\boldsymbol{\Sigma}\\ \boldsymbol{\Sigma}' &= \boldsymbol{\Sigma} - \frac{\boldsymbol{\Sigma}\mathbf{v}^T\mathbf{v}\boldsymbol{\Sigma}}{\sigma_2}.\end{aligned}$$

If $\sigma_2 = 0$, the conditional distribution of $\mathbf{X}\Big|(Y = y)$ is $\mathcal{D}^d_{\boldsymbol{\Sigma},\boldsymbol{\mu}}$.

Remark 8. We note that Corollary 7 is also valid for $\mathbf{X}|\langle\mathbf{X},\mathbf{v}\rangle$ by letting $\sigma_e = 0$.

2.3 Lattices

A *lattice*, denoted as $\varLambda$, is a discrete additive subgroup of $\mathbb{R}^m$, which is generated as the set of all linear integer combinations of n $(m \geq n)$ linearly independent basis vectors $\{\mathbf{b}_j\} \subset \mathbb{R}^m$, namely,

$$\varLambda := \left\{\sum\nolimits_j z_j\mathbf{b}_j : z_j \in \mathbb{Z}\right\},$$

We say that m is the *dimension* of $\varLambda$ and n is its rank. A lattice is *full rank* if $n = m$. A matrix $\mathbf{B}$ having the basis vectors as rows is called a *basis*. The *volume* of a lattice $\varLambda$ is defined as $\mathrm{Vol}(\varLambda) := \sqrt{\det(\mathbf{B}\mathbf{B}^T)}$. The *dual lattice* of $\varLambda$ in $\mathbb{R}^n$ is defined as follows.

$$\varLambda^* := \{\mathbf{y} \in \mathrm{Span}(\mathbf{B}) \mid \forall\mathbf{x} \in \varLambda, \langle\mathbf{x},\mathbf{y}\rangle \in \mathbb{Z}\}.$$

Note that, $(\varLambda^*)^* = \varLambda$, and $\mathrm{Vol}(\varLambda^*) = 1/\mathrm{Vol}(\varLambda)$.

Lemma 9 ([29, Proposition 1.3.4]). *Let $\varLambda$ be a lattice and let F be a subspace of $\mathbb{R}^n$. If $\varLambda \cap F$ is a lattice, then the dual of $\varLambda \cap F$ is the orthogonal projection onto F of the dual of $\varLambda$. In other words, each element of $\varLambda^*$ is multiplied by the projection matrix $\boldsymbol{\Pi}_F$:*

$$(\varLambda \cap F)^* = \varLambda^* \cdot \boldsymbol{\Pi}_F.$$

Lemma 10 ([29, Proposition 1.2.9]). *Let $\varLambda$ be a lattice in $\mathbb{R}^n$, let F be a subspace of $\mathbb{R}^n$ such that $\varLambda \cap F$ is a latticeand let $\boldsymbol{\Pi}_F^{\perp}$ be the orthogonal projection onto $F^{\perp}$. Then*

$$\mathrm{Vol}(\varLambda \cdot \boldsymbol{\Pi}_F^{\perp}) = \mathrm{Vol}(\varLambda)(\mathrm{Vol}(\varLambda \cap F)^{-1}).$$

Definition 11 (Primitive vectors). *A set of vector* $\mathbf{y}_1, \ldots, \mathbf{y}_k \in \Lambda$ *is said primitive with respect to* Λ *if* $\Lambda \cap \mathrm{Span}(\mathbf{y}_1, \ldots, \mathbf{y}_k)$ *is equal to the lattice generated by* $\mathbf{y}_1, \ldots, \mathbf{y}_k$*. Equivalently, it is primitive if it can be extended to a basis of* Λ*. If* $k = 1$*,* $\mathbf{y}_1$*, this is equivalent to* $\mathbf{y}_1/i \notin \Lambda$ *for any integer* $i \geq 2$*.*

To predict the hardness of the lattice reduction on altered instances , we must compute the volume of the final transformed lattice. We devise a highly efficient way to do this, by observing that each time a hint is integrated, we can update the volume of the transformed lattice, given only the volume of the previous lattice and information about the current hint (under mild restrictions on the form of the hint). In the following, we present lemmas that will be useful for progressively computing the volume of the transformed lattice when different types of hints are integrated.

Lemma 12 (Volume of a lattice slice). *Given a lattice* Λ *with volume* $\mathrm{Vol}(\Lambda)$*, and a primitive vector* $\mathbf{v}$ *with respect to* Λ^**. Let* $\mathbf{v}^\perp$ *denote subspace orthogonal to* $\mathbf{v}$*. Then* $\Lambda \cap \mathbf{v}^\perp$ *is a lattice with volume* $\mathrm{Vol}(\Lambda \cap \mathbf{v}^\perp) = \|\mathbf{v}\| \cdot \mathrm{Vol}(\Lambda)$*.*

Proof. Let use denote $\Lambda' = (\Lambda \cap \mathbf{v}^\perp) = \{\mathbf{x} \in \Lambda \mid \langle \mathbf{x}, \mathbf{v} \rangle = 0\}$. We now compute $\mathrm{Vol}(\Lambda')$ as follows

$$\begin{aligned}
\mathrm{Vol}(\Lambda') = \frac{1}{\mathrm{Vol}(\Lambda'^*)} &= \frac{1}{\mathrm{Vol}(\Lambda^* \cdot \mathbf{\Pi}_{\mathbf{v}}^\perp)} && (1) \\
&= \frac{\mathrm{Vol}\left(\Lambda^* \cap \mathrm{Span}(\mathbf{v})\right)}{\mathrm{Vol}(\Lambda^*)} && (2) \\
&= \mathrm{Vol}\left(\Lambda^* \cap \mathrm{Span}(\mathbf{v})\right) \mathrm{Vol}(\Lambda),
\end{aligned}$$

where Equation (1) follows from Lemma 9, and Equation (2) follows from Lemma 10. By Definition 11, $\mathbf{v}$ generates the one-dimensional lattice $\Lambda^* \cap \mathrm{Span}(\mathbf{v})$, and $\mathrm{Vol}(\Lambda^* \cap \mathrm{Span}(\mathbf{v})) = \|\mathbf{v}\|$. Therefore we have $\mathrm{Vol}(\Lambda') = \|\mathbf{v}\| \cdot \mathrm{Vol}(\Lambda)$.

Lemma 13 (Volume of a sparsified lattice). *Let* Λ *be a lattice,* $\bar{\mathbf{v}} \in \Lambda^*$ *be a primitive vector of* Λ^**, and* $k > 0$ *be an integer. Let* $\Lambda' = \{\mathbf{x} \in \Lambda \mid \langle \mathbf{x}, \bar{\mathbf{v}} \rangle = 0 \bmod k\}$ *be a sublattice of* Λ*. Then* $\mathrm{Vol}(\Lambda') = k \cdot \mathrm{Vol}(\Lambda)$*.*

Proof. Because $\bar{\mathbf{v}}$ is a dual vector of Λ, we have $\langle \bar{\mathbf{v}}, \Lambda \rangle \subset \mathbb{Z}$. Let ℓ be such that, $\langle \bar{\mathbf{v}}, \Lambda \rangle = \ell\mathbb{Z}$. Note that $\bar{\mathbf{v}}/\ell \in \Lambda^*$, therefore, by primitivity of $\bar{\mathbf{v}}$, we have $\ell = 1$. In particular, the group morphism $\phi : \mathbf{x} \in \Lambda \mapsto \langle v, \Lambda \rangle \bmod k$ is surjective. Note that $\Lambda' = \ker \phi$, therefore we have $|\Lambda/\Lambda'| = |\mathbb{Z}_k| = k$. We conclude.

Fact 14 (Volume of a projected lattice) *Let* Λ *be a lattice,* $\mathbf{v} \in \Lambda$ *be a primitive vector of* Λ*. Let* $\Lambda' = \Lambda \cdot \mathbf{\Pi}_{\mathbf{v}}^\perp$ *be a sublattice of* Λ*. Then* $\mathrm{Vol}(\Lambda') = \mathrm{Vol}(\Lambda)/\|\mathbf{v}\|$*. More generally, if* $\mathbf{V}$ *is a primitive set of vectors of* Λ*, then* $\Lambda' = \Lambda \cdot \mathbf{\Pi}_{\mathbf{V}}^\perp$ *has volume* $\mathrm{Vol}(\Lambda') = \mathrm{Vol}(\Lambda)/\sqrt{\det(\mathbf{V}\mathbf{V}^T)}$*.*

Fact 15 (Lattice volume under linear transformations) *Let* Λ *be a lattice in* $\mathbb{R}^n$*, and* $\mathbf{M} \in \mathbb{R}^{n \times n}$ *a matrix such that* $\ker \mathbf{M} = \mathrm{Span}(\Lambda)^\perp$*. Then we have* $\mathrm{Vol}(\Lambda \cdot \mathbf{M}) = \mathrm{rdet}(\mathbf{M}) \, \mathrm{Vol}(\Lambda)$*.*

3 Distorted Bounded Distance Decoding

3.1 Definition

We first recall the definition of the (search) LWE problem, in its short-secret variant which is the most relevant to practical LWE-based encryption.

Definition 16 (Search LWE problem with short secrets.). *Let n, m and q be positive integers, and let χ be a distribution over $\mathbb{Z}$. The* search LWE problem (with short secrets) *for parameters (n, m, q, χ) is:*

***Given** the pair $\left(\mathbf{A} \in \mathbb{Z}_q^{m\times n}, \mathbf{b} = \mathbf{z}\mathbf{A}^T + \mathbf{e} \in \mathbb{Z}_q^m\right)$ where:*

1. *$\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ is sampled uniformly at random,*
2. *$\mathbf{z} \leftarrow \chi^n$, and $\mathbf{e} \leftarrow \chi^m$ are sampled with independant and identically distributed coefficients following the distribution χ.*

***Find** $\mathbf{z}$.*

The primal attack (See for example [4]) against (search)-LWE proceeds by viewing the LWE instance as an instance of a Bounded Distance Decoding (BDD) problem, converting it to a uSVP instance (via Kannan's embedding [24]), and finally applying a lattice reduction algorithm to solve the uSVP instance. The central tool of our framework is a generalization of BDD that accounts for potential distortion in the distribution of the secret noise vector that is to be recovered.

Definition 17 (Distorted Bounded Distance Decoding problem). *Let $\Lambda \subset \mathbb{R}^d$ be a lattice, $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$ be a symmetric matrix and $\boldsymbol{\mu} \in \mathrm{Span}(\Lambda) \subset \mathbb{R}^d$ such that*

$$\mathrm{Span}(\mathbf{\Sigma}) \subsetneq \mathrm{Span}(\mathbf{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}) = \mathrm{Span}(\Lambda). \tag{3}$$

The Distorted Bounded Distance Decoding problem $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\mathbf{\Sigma}}$ is the following problem:

***Given** $\boldsymbol{\mu}, \mathbf{\Sigma}$ and a basis of Λ.*
***Find** the unique vector $\mathbf{x} \in \Lambda \cap E(\boldsymbol{\mu}, \mathbf{\Sigma})$*

where $E(\boldsymbol{\mu}, \mathbf{\Sigma})$ denotes the ellipsoid

$$E(\boldsymbol{\mu}, \mathbf{\Sigma}) := \{\mathbf{x} \in \boldsymbol{\mu} + \mathrm{Span}(\mathbf{\Sigma}) | (\mathbf{x} - \boldsymbol{\mu}) \cdot \mathbf{\Sigma}^{\sim} \cdot (\mathbf{x} - \boldsymbol{\mu})^T \leq \mathrm{rank}(\mathbf{\Sigma})\}.$$

We will refer to the triple $\mathcal{I} = (\Lambda, \boldsymbol{\mu}, \mathbf{\Sigma})$ as the instance of the $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\mathbf{\Sigma}}$ problem.

Intuitively, Definition 17 corresponds to knowing that the secret vector $\mathbf{x}$ to be recovered follows a distribution of variance $\mathbf{\Sigma}$ and average $\boldsymbol{\mu}$. The quantity $(\mathbf{x} - \boldsymbol{\mu}) \cdot \mathbf{\Sigma}^{\sim} \cdot (\mathbf{x} - \boldsymbol{\mu})^T$ can be interpreted as a non-canonical Euclidean squared distance $\|\mathbf{x} - \boldsymbol{\mu}\|_{\mathbf{\Sigma}}^2$, and the expected value of such a distance for a Gaussian $\mathbf{x}$ of variance $\mathbf{\Sigma}$ and average $\boldsymbol{\mu}$ is $\mathrm{rank}(\mathbf{\Sigma})$. One can argue that, for such a Gaussian, there is a constant probability that $\|\mathbf{x} - \boldsymbol{\mu}\|_{\mathbf{\Sigma}}^2$ is slightly greater than $\mathrm{rank}(\mathbf{\Sigma})$. Since we are interested in the average behavior of our attack, we ignore this benign technical detail. In fact, we will typically interpret DBDD as the promise that the secret follows a Gaussian distribution of center $\boldsymbol{\mu}$ and covariance $\mathbf{\Sigma}$.

The ellipsoid can be seen as an affine transformation (that we call "distortion") of the centered hyperball of radius $\mathrm{rank}(\mathbf{\Sigma})$. Let us introduce a notation for the hyperball; for any $d \in \mathbb{N}$

$$B_d := \{\mathbf{x} \in \mathbb{R}^{d\times d} \mid \|\mathbf{x}\|_2 \leq d\}. \tag{4}$$

One can thus write using Definition 2:

$$E(\boldsymbol{\mu}, \mathbf{\Sigma}) = B_{\mathrm{rank}(\mathbf{\Sigma})} \cdot \sqrt{\mathbf{\Sigma}} + \boldsymbol{\mu}. \tag{5}$$

From the Span inclusion in Equation (3), one can deduce that the condition is equivalent to requiring $\boldsymbol{\mu} \notin \mathrm{Span}(\mathbf{\Sigma})$ and $\mathrm{rank}(\mathbf{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}) = \mathrm{rank}(\mathbf{\Sigma}) + 1 = \mathrm{rank}(\Lambda)$. This detail is necessary for embedding it properly into a uSVP instance (See later in Section 3.3).

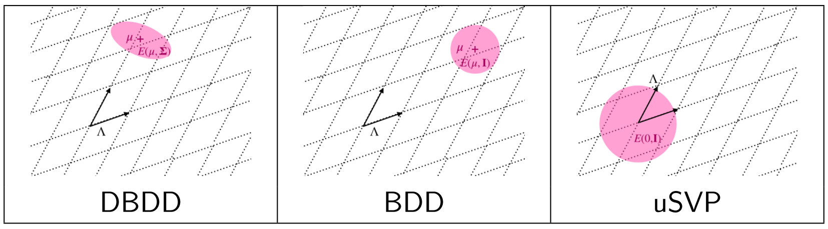


Fig. 3. Graphical intuition of DBDD, BDD and uSVP in dimension two: the problem consists in finding a nonzero element of Λ in the colored zone. The identity hyperball is larger for uSVP to represent the fact that, during the reduction, the uSVP lattice has one dimension more than for BDD.

Particular cases of Definition 17. Let us temporarily ignore the condition in Equation (3) to study some particular cases. As shown in Figure 3, when $\mathbf{\Sigma} = \mathbf{I}_d$, $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\mathbf{I}_d}$ is BDD instance. Indeed, the ellipsoid becomes a shifted hyperball $E(\boldsymbol{\mu}, \mathbf{I}_d) = \{\mathbf{x} \in \boldsymbol{\mu} + \mathbb{R}^{d\times d} \mid \|\mathbf{x} - \boldsymbol{\mu}\|_2 \leq d\} = B_d + \boldsymbol{\mu}$. If in addition $\boldsymbol{\mu} = 0$, $\mathsf{DBDD}_{\Lambda,\mathbf{0},\mathbf{I}_d}$ becomes a uSVP instance on Λ.

3.2 Embedding LWE into DBDD

In the typical primal attack framework (Figure 1), one directly views LWE as a BDD instance of the same dimension. For our purposes, however, it will be useful to apply Kannan's Embedding at this stage and therefore increase the dimension of the lattice by 1. While it could be delayed to the last stage of our attack, this extra fixed coefficient 1 will be particularly convenient when we integrate hints (see Remark 23 in Section 4). It should be noted that no information is lost through this transformation, since the parameters $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ allow us to encode the knowledge that the solution we are looking for has its last coefficient set to 1 and nothing else.

In more details, the solution $\mathbf{s} := (\mathbf{e}, \mathbf{z})$ of an LWE instance is extended to

$$\bar{\mathbf{s}} := (\mathbf{e}, \mathbf{z}, 1) \tag{6}$$

which is a short vector in the lattice $\Lambda = \left\{(\mathbf{x}, \mathbf{y}, w) \,|\mathbf{x} + \mathbf{y}\mathbf{A}^{\mathrm{T}} - \mathbf{b}w = 0 \mod q\right\}$. A basis of this lattice is given by the row vectors of

$$\begin{bmatrix} q\mathbf{I}_m & 0 & 0 \\ \mathbf{A}^{\mathrm{T}} & -\mathbf{I}_n & 0 \\ \mathbf{b} & 0 & c \end{bmatrix}.$$

Denoting μ_χ and σ_χ^2 the average and variance of the LWE distribution χ (See Definition 16), we can convert this LWE instance to a $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\mathbf{\Sigma}}$ instance with $\boldsymbol{\mu} = [\mu_\chi \cdots \mu_\chi \; 1]$ and $\mathbf{\Sigma} = \left[\begin{smallmatrix} \sigma_\chi^2 \mathbf{I}_{m+n} & 0 \\ 0 & 0 \end{smallmatrix}\right]$. The lattice Λ is of full rank in $\mathbb{R}^d$ where $d := m + n + 1$, and its volume is q^m. Note that the rank of $\mathbf{\Sigma}$ is only $d - 1$: the ellipsoid has one less dimension than the lattice. It then validates the requirement of Equation (3).

Remark 18. Typically, Kannan's embedding from BDD to uSVP leaves the bottom right matrix coefficient c as a free parameter to be chosen optimally. The optimal value is the one maximizing

$$\frac{\|(\mathbf{z}; c)\|}{\det(\Lambda)^{1/d}} = \frac{(m+n)\sigma_\chi + c}{(c \cdot q^m)^{1/d}},$$

namely, $c = \sigma_\chi$ according to the arithmetic-geometric mean inequality. Some prior works [4,6] instead chose $c = 1$. While this is benign since σ_χ is typically not too far from 1, it remains a sub-optimal choice. Looking ahead, in our DBDD framework, this choice becomes irrelevant thanks to the *isotropization* step introduced in the next section; we can therefore choose any value without affecting the final result.

3.3 Converting DBDD to uSVP

In this Section, we explain how a DBDD instance $(\Lambda, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is converted into a uSVP one. Two modifications are necessary.

First, let us show that the ellipsoid in Definition 17 is contained in a larger centered ellipsoid (with one more dimension) as follows:

$$E(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \subset E(\mathbf{0}, \boldsymbol{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}). \tag{7}$$

Using Equation (5), one can write

$$E(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = B_{\mathrm{rank}(\boldsymbol{\Sigma})} \cdot \sqrt{\boldsymbol{\Sigma}} + \boldsymbol{\mu} \subset B_{\mathrm{rank}(\boldsymbol{\Sigma})} \cdot \sqrt{\boldsymbol{\Sigma}} \pm \boldsymbol{\mu},$$

where $B_{\mathrm{rank}(\boldsymbol{\Sigma})}$ is defined in Equation (4). And, with Equation (3), one can deduce $\mathrm{rank}(\boldsymbol{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}) = \mathrm{rank}(\boldsymbol{\Sigma}) + 1$, then:

$$B_{\mathrm{rank}(\boldsymbol{\Sigma})} \cdot \sqrt{\boldsymbol{\Sigma}} \pm \boldsymbol{\mu} \subset B_{\mathrm{rank}(\boldsymbol{\Sigma})+1} \cdot \begin{bmatrix} \sqrt{\boldsymbol{\Sigma}} \\ \boldsymbol{\mu} \end{bmatrix}.$$

We apply Definition 2 which confirms the inclusion of Equation (7):

$$E(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \subset B_{\mathrm{rank}(\boldsymbol{\Sigma})+1} \cdot \begin{bmatrix} \sqrt{\boldsymbol{\Sigma}} \\ \boldsymbol{\mu} \end{bmatrix} = E(\mathbf{0}, \boldsymbol{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}).$$

Thus, we can homogenize and transform the instance into a centered one with $\boldsymbol{\Sigma}' := \boldsymbol{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}$.

Secondly, to get an isotropic distribution (i.e. with all its eigenvalues being 1), one can just multiply every element of the lattice with the pseudoinverse of $\sqrt{\boldsymbol{\Sigma}'}$. We get a new covariance matrix $\boldsymbol{\Sigma}'' = \sqrt{\boldsymbol{\Sigma}'}^{\sim} \cdot \boldsymbol{\Sigma}' \cdot \sqrt{\boldsymbol{\Sigma}'}^{\sim T} = \boldsymbol{\Pi}_{\boldsymbol{\Sigma}'} \cdot \boldsymbol{\Pi}_{\boldsymbol{\Sigma}'}{}^T$. And since $\boldsymbol{\Pi}_{\boldsymbol{\Sigma}'} = \boldsymbol{\Pi}_{\boldsymbol{\Sigma}'}^T$ and $\boldsymbol{\Pi}_{\boldsymbol{\Sigma}'}^2 = \boldsymbol{\Pi}_{\boldsymbol{\Sigma}'}$ (see Section 2.1), $\boldsymbol{\Sigma}'' = \boldsymbol{\Pi}_{\boldsymbol{\Sigma}'} = \boldsymbol{\Pi}_{\Lambda}$, the last equality coming from Equation (3).
In a nutshell, one must make by the two following changes:

$$\begin{aligned} \text{homogenize: } & (\Lambda, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \mapsto (\Lambda, \mathbf{0}, \boldsymbol{\Sigma}' := \boldsymbol{\Sigma} + \boldsymbol{\mu}^T \cdot \boldsymbol{\mu}) \\ \text{isotropize: } & (\Lambda, \mathbf{0}, \boldsymbol{\Sigma}') \mapsto (\Lambda \cdot \mathbf{M}, \mathbf{0}, \boldsymbol{\Pi}_{\Lambda}) \end{aligned}$$

where $\mathbf{M} := (\sqrt{\boldsymbol{\Sigma}'})^{\sim}$.
From the solution $\mathbf{x}$ to the $\mathsf{uSVP}_{\Lambda \cdot \mathbf{M}}$ problem, one can derive $\mathbf{x}' = \mathbf{x}\mathbf{M}^{\sim}$ the solution to the $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\boldsymbol{\Sigma}}$ problem. Note that, to solve a DBDD instance, one can ignore the isotropization and directly apply lattice reduction before the second step. This leads, however, to less efficient attacks.

Remark 19. One should note that the first homogenization step “forgets” some information about the secret’s distribution. This, however, is inherent to the conversion to a unique-SVP problem which is geometrically homogeneous, and is already present in the original primal attack.

3.4 Security estimates of uSVP: bikz versus bits

The attack on a uSVP instance consists of applying BKZ-β on the uSVP lattice Λ for an appropriate block size parameter β. The cost of the attack grows with β, however, modeling this cost precisely is at the moment rather delicate, as the state of the art seems to still be in motion. Numerous NIST candidates choose to underestimate this cost, keeping a margin to accommodate future improvements, and there seems to be no clear consensus on which model to use (see [1] for a summary of existing cost models).

While this problem is orthogonal to our work, we still wish to be able to formulate quantitative security losses. We therefore express all concrete security estimates using the blocksize β as our measure of the level of security, and treat the latter as a measurement of the security level in a unit called the *bikz*. We thereby leave the question of the exact bikz-to-bit conversion estimate outside the scope of this paper, and recall that those conversion formulae are not necessarily linear, and may have small dependency in other parameters. For the sake of concreteness, we note that certain choose, for example, to claim 128 bits of security for 380 bikz, and in this range, most models suggest a security increase of one bit every 2 to 4 bikz.

Remark 20. We also clarify that the estimates given in this paper only concern the pure lattice attack via the uSVP embedding discussed above. In particular, we note that some NIST candidates with ternary secrets [28] also consider the hybrid attack of [23], which we ignore in this work. We nevertheless think that the compatibility with our framework is plausible, with some effort.

Predicting β from a uSVP instance The state-of-the-art predictions for solving uSVP instances using BKZ were given in [5,4]. Namely, for $\varLambda$ a lattice of dimension $\dim(\varLambda)$, it is predicted that BKZ-β can solve a $\mathsf{uSVP}_\varLambda$ instance with secret $\mathbf{s}$ when

$$\sqrt{\beta/\dim(\varLambda)} \cdot \|\mathbf{s}\| \leq \delta_\beta^{2\beta-\dim(\varLambda)-1} \cdot \mathrm{Vol}(\varLambda)^{1/\dim(\varLambda)} \tag{8}$$

where δ_β is the so called root-Hermite-Factor of BKZ-β. For $\beta \geq 50$, the Root-Hermite-Factor is predictable using the Gaussian Heuristic [13]:

$$\delta_\beta = \left((\pi\beta)^{\frac{1}{\beta}} \cdot \frac{\beta}{2\pi e} \right)^{1/(2\beta-2)}. \tag{9}$$

Note that the uSVP instances we generate are isotropic and centered so that the secret has covariance $\mathbf{\Sigma} = \mathbf{I}$ (or $\mathbf{\Sigma} = \mathbf{\Pi}_\varLambda$ if $\varLambda$ is not of full rank) and $\mu = \mathbf{0}$. Thus, on average, we have $\|\mathbf{s}\|^2 = \mathrm{rank}(\mathbf{\Sigma}) = \dim(\varLambda)$. Therefore, β can be estimated as the minimum integer that satisfies

$$\sqrt{\beta} \leq \delta_\beta^{2\beta-\dim(\varLambda)-1} \cdot \mathrm{Vol}(\varLambda)^{1/\dim(\varLambda)}. \tag{10}$$

While β must be an integer as a BKZ parameter, we nevertheless provide a continuous value, for a finer comparison of the difficulty of an instance.

Remark 21. To predict security, one does not need the basis of $\varLambda$, but only its dimension and its volume. Similarly, it is not necessary to explicitly compute the isotropization matrix $\mathbf{M}$ of Section 3.3, thanks to Fact 15: $\mathrm{Vol}(\varLambda \cdot \mathbf{M}) = \mathrm{rdet}(\mathbf{M})\,\mathrm{Vol}(\varLambda) = \mathrm{rdet}(\mathbf{\Sigma}')^{-1/2}\,\mathrm{Vol}(\varLambda)$. These two shortcuts will allow us to efficiently make predictions for cryptographically large instances, in our *lightweight* implementation of Section 5.

Refined prediction for small blocksizes The work of [4] warns about a regime where those predictions are not accurate, due to a so-called *second-intersection* between the predicted lengths of the Gram-Schmidt vectors and the successive projections of the secret. This phenomenon only appears for small blocksizes β, which is not relevant for cryptographically hard instances. However, we would still like to be able to make reliable predictions for small blocksizes as well, so as to test the validity of our predictions with and without hints.

Other sources of inaccuracy of this model are the so-called head and tails phenomenon [38,7], as well as the fact that one can be lucky: the projected length of the secret can vary, making it plausible that the secret will be found with a slightly smaller blocksize. For example, in [4] more than 50% of the attacks were already successful by running BKZ with blocksize $\beta_{\text{pred}} - 5$.

Furthermore, the predictions of [4] work under the assumption that as soon as the projected secret vector has been detected at position $d - \beta$, it will be "pulled-back" to the front by the run of LLL that is typically executed between BKZ tours. For large block-sizes β this event is indeed very likely as argued and experimentally verified in [4], but may not occur in small or intermediate dimension. In fact, the issue of double-intersection is precisely related to this assumption.

For experimental validation purposes of our work, we prefer to have accurate prediction even for small blocksizes. We therefore devise a refined strategy. First, we resort to the so called BKZ-simulator [13] to predict more accurately the length ℓ_i of the Gram-Schmidt vectors. Secondly, we do not assume that the projected secret $\pi_i(\mathbf{s})$ (projected orthogonally to the $i-1$ first vectors of the reduced basis, as in [4]) has exactly length $\sqrt{n-i}$, but simply treat it as a spherical Gaussian. We can therefore compute the probability that it is detected at position i by considering the CDF of χ^2_{n-i}, the chi-square distribution with $n - i$ degrees of freedom.

At last, we do not only account for the detectability of the secret vector at position $i = n-\beta$, but also check whether it is likely that the vector will be pulled to the front (not by the interleaved LLL, by BKZ itself, which is more powerful). That is, we consider the probability that:

$$E_i : \|\pi_i(\mathbf{s})\| \leq \ell_i \text{ simultaneously for all } i \in \{d-\beta, d-2\beta+1, d-3\beta+2, \dots\}.$$

Those events are not perfectly independent, which makes computing the probability of the conjunction of those more painful.[7] For simplicity, we only account for dependence between consecutive events E_i and E_{i+1} and therefore avoid having to resort to numerical computation of nested integrals. We iteratively compute the success probability for each tour of BKZ-β for increasing β, and from there deduce the average successful β.

As depicted in Figure 4, our new methodology leads to much more satisfactory estimates compared to the model from the literature [4,5]. In particular, for low blocksize the literature widely underestimates the required blocksize, which is due to only considering detectability at position $d - \beta$. For large blocksize, it somewhat overestimates it, which could be attributed to the fact that it does not account for luck. On the contrary, our new methodology seems quite precise in all regimes, making errors of at most 1 bikz. This new methodology certainly deserves further study and refinement, which we leave to future work.

4 Including hints

In this Section, we define several categories of hints—**perfect hints**, **modular hints**, **approximate hints** (**conditioning** and ***a posteriori***), and **short vector hints**—and show that these types of hints can be integrated into a DBDD instance. Hints belonging to these categories typically have the form of a linear equation in $\mathbf{s}$ (and possibly additional variables). As emphasized in Section 1, these hints have lattice-friendly forms and their usefulness in realistic applications may not be obvious. We refer to Section 6 for detailed applications of these hints.

The technical challenge, therefore, is to characterize the effect of such hints on the DBDD instance—i.e. determine the resulting $(\Lambda', \boldsymbol{\mu}', \boldsymbol{\Sigma}')$ of the new DBDD instance, after the hint is incorporated.

Henceforth, let $\mathcal{I} = \mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\boldsymbol{\Sigma}}$ be a fixed instance and let $\mathbf{s}$ be its secret solution. Each hint will introduce new constraints on $\mathbf{s}$ and will ultimately decrease the security level.

[7] The expert reader may note that, for $\mathbf{s}$ uniformly distributed over a sphere, such conjunction correspond to a cylinder interesection, as used for pruning in enumeration [18].

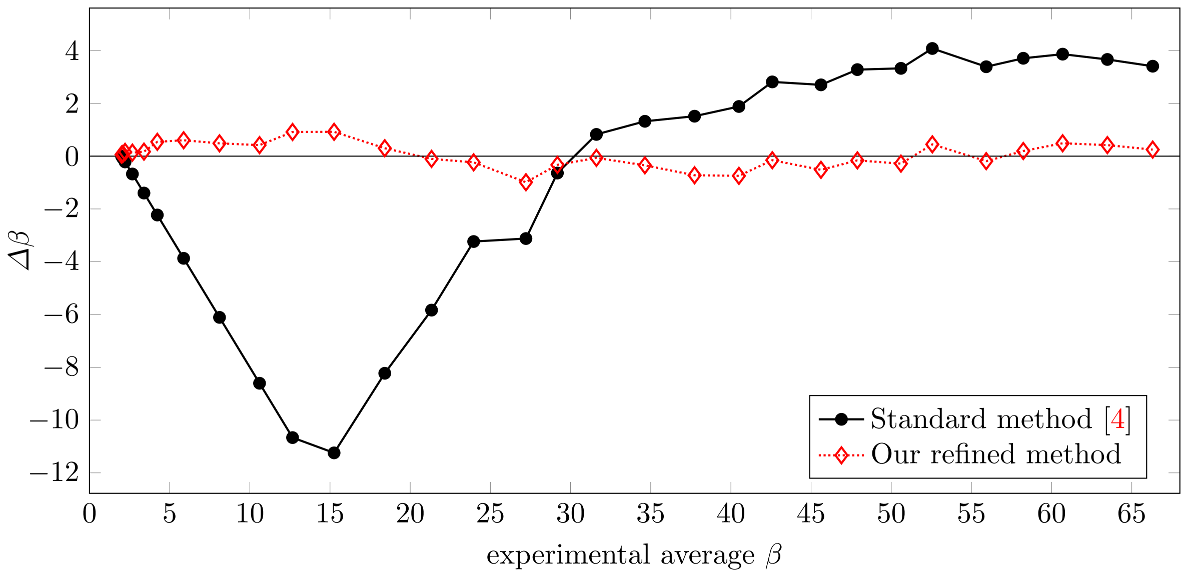


Fig. 4. The difference $\Delta\beta = \text{real}-\text{predicted}$, as a function of the average experimental beta β. The experiment consists in running a single tour of BKZ-β for $\beta = 2, 3, 4, \ldots$ until the secret short vector is found. This was averaged over 256 many LWE instances per data-point, for parameters $q = 3301$, $\sigma = 20$ and $n = m \in \{30, 32, 34, \ldots, 88\}$.

Non-Commutativity It should be noted that many types of hints commute: Integrating them in any order will lead to the same DBDD instance. Potential exceptions are **non-smooth modular hints** (See later in Section 4.2) and ***aposteriori* approximate hints** (See later in Section 4.4): they do not always commute with the other types of hints, and do not always commute between themselves, unless the vectors $\mathbf{v}$'s of those hints are all orthogonal to each other. The reason is: in these cases, the distribution in the direction of $\mathbf{v}$ is redefined which erases the prior information.

4.1 Perfect Hints

Definition 22 (Perfect hint). *A perfect hint on the secret* $\mathbf{s}$ *is the knowledge of* $\mathbf{v} \in \mathbb{Z}^d$ *and* $l \in \mathbb{Z}$*, such that*

$$\langle \mathbf{s},\ \mathbf{v} \rangle = l,$$

A perfect hint is quite strong in terms of additional knowledge. It allows decreasing the dimension of the lattice by one. One could expect such hints to arise from the following scenarios:

– The full leak without noise of an original coefficient, or even an unreduced intermediate register since most of the computations are linear. For the second case, one may note that optimized implementations of NTT typically attempt to delay the first reduction modulo q, so leaking a register on one of the first few levels of the NTT would indeed lead to such a hint.
– A noisy leakage of the same registers, but with still a rather high guessing confidence. In that case it may be worth making the guess while decreasing the success probability of the attack.[8] This could happen in a cold-boot attack scenario. This is also the case in the single trace attack on Frodo [11] that we will study as one of our examples in Section 6.1.
– More surprisingly, certain schemes, including some NIST candidates offer such a hint 'by design'. Indeed, LAC, Round5 and NTRU-HPS all choose ternary secret vectors with a

[8] One may then re-amplify the success probability by retrying the attack making guesses at different locations

prescribed number of 1's and -1's, which directly induce one or two such perfect hints. This will be detailed in Section 6.3.

Integrating a perfect hint into a DBDD instance Let $\mathbf{v} \in \mathbb{Z}^d$ and $l \in \mathbb{Z}$ be such that $\langle \mathbf{s}, \mathbf{v} \rangle = l$. Note that the hint can also be written as

$$\langle \bar{\mathbf{s}},\ \bar{\mathbf{v}} \rangle = 0,$$

where $\bar{\mathbf{s}}$ is the extended LWE secret as defined in Equation (6) and $\bar{\mathbf{v}} := (\mathbf{v}\,;\,-l)$.

Remark 23. Here we understand the interest of using Kannan's embedding *before* integrating hints rather than after: it allows to also homogenize the hint, and therefore to make Λ' a proper lattice rather than a lattice coset (i.e. a shifted lattice).

Including this hint is done by modifying the $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\boldsymbol{\Sigma}}$ to $\mathsf{DBDD}_{\Lambda',\boldsymbol{\mu}',\boldsymbol{\Sigma}'}$, where:

$$\Lambda' = \Lambda \cap \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \langle \mathbf{x}, \bar{\mathbf{v}} \rangle = 0 \right\}$$

$$\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} - \frac{(\bar{\mathbf{v}}\boldsymbol{\Sigma})^T \bar{\mathbf{v}}\boldsymbol{\Sigma}}{\bar{\mathbf{v}}\boldsymbol{\Sigma}\bar{\mathbf{v}}^T} \tag{11}$$

$$\boldsymbol{\mu}' = \boldsymbol{\mu} - \frac{\langle \bar{\mathbf{v}}, \boldsymbol{\mu} \rangle}{\bar{\mathbf{v}}\boldsymbol{\Sigma}\bar{\mathbf{v}}^T} \bar{\mathbf{v}}\boldsymbol{\Sigma} \tag{12}$$

We now explain how to derive the new mean $\boldsymbol{\mu}'$ and the new covariance $\boldsymbol{\Sigma}'$. Let Y be the random variable $\langle \bar{\mathbf{s}}, \bar{\mathbf{v}} \rangle$, where $\bar{\mathbf{s}}$ has mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Then $\boldsymbol{\mu}'$ is the mean of $\bar{\mathbf{s}}$ conditioned on $Y = 0$, and $\boldsymbol{\Sigma}'$ is the covariance of $\bar{\mathbf{s}}$ conditioned on $Y = 0$. Using Corollary 7, we obtain the corresponding conditional mean and covariance.

We note that lattice Λ' is an intersection of Λ and a hyperplane orthogonal to $\bar{\mathbf{v}}$. Given $\mathbf{B}$ as basis of Λ, by [30, Theorem 5], a basis of Λ' can be computed as follows:

1. Let $\mathbf{D}$ be dual basis of $\mathbf{B}$. Compute $\mathbf{D}_\perp := \mathbf{D} \cdot \boldsymbol{\Pi}_{\bar{\mathbf{v}}}^\perp$.
2. Apply LLL algorithm on $\mathbf{D}_\perp$ to eliminate linear dependencies. Then delete the first row of $\mathbf{D}_\perp$ (which is $\mathbf{0}$ because with the hyperplane intersection, the dimension of the lattice is decremented).
3. Output the dual of the resulting matrix.

While polynomial time, the above computation is quite heavy, especially as there is no convenient library offering a parallel version of LLL. Fortunately, for predicting attack costs, one only needs the dimension of the lattice Λ and its volume. These can easily be computed assuming $\bar{\mathbf{v}}$ is a primitive vector (See Definition 11) of the dual lattice: the dimension decreases by 1, and the volume increases by a factor $||\bar{\mathbf{v}}||$. This is stated and proved in Lemma 12. Intuitively, the primitivity condition is needed since then one can scale the leak to $\langle \mathbf{s}, f\mathbf{v} \rangle = fl$ for any non-zero factor $f \in \mathbb{R}$ and get an equivalent leak; however there is only one factor f that can ensure that $f\bar{\mathbf{v}} \in \Lambda^*$, and is primitive in it.

Remark 24. Note that if $\bar{\mathbf{v}}$ is not in the span of Λ—as typically occurs if other non-orthogonal perfect hints have already been integrated—Lemma 12 should be applied to the orthogonal projection $\bar{\mathbf{v}}' = \bar{\mathbf{v}} \cdot \boldsymbol{\Pi}_\Lambda$ of $\bar{\mathbf{v}}$ onto Λ. Indeed, the perfect hint $\langle \bar{\mathbf{s}},\ \bar{\mathbf{v}}' \rangle = 0$ replacing $\bar{\mathbf{v}}$ by $\bar{\mathbf{v}}'$ is equally valid.

4.2 Modular Hints

Definition 25 (Modular hint). *A modular hint on the secret* $\mathbf{s}$ *is the knowledge of* $\mathbf{v} \in \mathbb{Z}^d$*,* $k \in \mathbb{Z}$ *and* $l \in \mathbb{Z}$*, such that*

$$\langle \mathbf{s},\ \mathbf{v} \rangle = l \mod k.$$

We can expect such hints to arise from several scenarios:

- obtaining the value of an intermediate register during LWE decryption would likely correspond to giving such a modular equation modulo q. This is also the case if an NTT coefficient leaks in a Ring-LWE scheme. It can also occur "by design" if the LWE secret is chosen so that certain NTT coordinates are fixed to 0 modulo q, as is the case in some instances of Order LWE [9].
- obtaining the absolute value $a = |s|$ of a coefficient s implies $s = a \bmod 2a$, and such a hint could be obtained by a timing attack on an unprotected implementation of a table-based sampler, in the spirit of [12].
- obtaining the Hamming weight of the string $b_1 b_2 \dots b'_1 b'_2 \dots$ used to sample a centered binomial coefficient $s = \sum b_i - \sum b'_i$ (as done in NewHope and Kyber [37,34]) reveals in particular $s \bmod 2$. Indeed, the latter string (or at least some parts of it) is more likely to be leaked than the Hamming weight of s.

Integrating a modular hint into a DBDD instance. Let $\mathbf{v} \in \mathbb{Z}^d$ and $l \in \mathbb{Z}$ be such that $\langle \mathbf{s}, \mathbf{v} \rangle = l \mod k$. Note that the hint can also be written as

$$\langle \bar{\mathbf{s}},\ \bar{\mathbf{v}} \rangle = 0 \mod k \tag{13}$$

where $\bar{\mathbf{s}}$ is the extended LWE secret as defined in Equation 6 and $\bar{\mathbf{v}} := (\mathbf{v}\,;\, -l)$. We refer to Remark 23 for the legitimacy of such dimension increase.

Intuitively, such a hint should only sparsify the lattice, and leave the average and the variance unchanged. This is not entirely true, this is only (approximately) true when the variance is sufficiently large in the direction of $\mathbf{v}$ to ensure smoothness, i.e. when $k^2 \ll \mathbf{v}\boldsymbol{\Sigma}\mathbf{v}^T$; one can refer to [31, Lemma 3.3 and Lemma 4.2] for the quality of that approximation. In this smooth case, we therefore have:

$$\Lambda' = \Lambda \cap \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \langle \mathbf{x}, \mathbf{v} \rangle = 0 \mod k \right\} \tag{14}$$

$$\boldsymbol{\mu}' = \boldsymbol{\mu} \tag{15}$$

$$\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} \tag{16}$$

On the other hand, if $k^2 \gg \mathbf{v}\boldsymbol{\Sigma}\mathbf{v}^T$, then the residual distribution will be highly concentrated on a single value, and one should therefore instead use a perfect $\langle \mathbf{s},\ \mathbf{v} \rangle = l + ik$ for some i.

Non-smooth case. In between, we call it the non-smooth case as the smoothness is not ensured and it is not a perfect hint. The lattice should then be sparsified as in the smooth case with Equation (14). One can resort to a numerical computation of the average μ_c and the variance σ_c^2 of the one-dimensional centered discrete Gaussian of variance $\sigma^2 = \mathbf{v}\boldsymbol{\Sigma}\mathbf{v}^T$ over the coset $l + k\mathbb{Z}$, and apply the corrections:

$$\boldsymbol{\mu}' = \boldsymbol{\mu} \cdot \boldsymbol{\Pi}_{\bar{\mathbf{v}}}^{\perp} + \mu_c \cdot \frac{\bar{\mathbf{v}}}{\|\bar{\mathbf{v}}\|^2} \tag{17}$$

$$\boldsymbol{\Sigma}' = (\boldsymbol{\Pi}_{\bar{\mathbf{v}}}^{\perp})^T \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{\Pi}_{\bar{\mathbf{v}}}^{\perp} + \sigma_c^2 \cdot \frac{\bar{\mathbf{v}}^T \cdot \bar{\mathbf{v}}}{\|\bar{\mathbf{v}}\|^4} \tag{18}$$

Intuitively, the projection $\mathbf{\Pi}^{\perp}_{\bar{\mathbf{v}}}$ completely erases prior information on $\langle \mathbf{s}, \bar{\mathbf{v}} \rangle$, before it is replaced by the new average and variance in the adequate direction.

As for perfect hints, the computation of Λ' can be done by working on the dual lattice. More specifically:

1. Let $\mathbf{D}$ be dual basis of $\mathbf{B}$.
2. Redefine $\bar{\mathbf{v}} \leftarrow \bar{\mathbf{v}} \cdot \mathbf{\Pi}_\Lambda$, noting that this does not affect the validity of the hint.
3. Append $\bar{\mathbf{v}}/(k\|\bar{\mathbf{v}}\|^2)$ to $\mathbf{D}$ and obtain $\mathbf{D}'$
4. Apply LLL algorithm on $\mathbf{D}'$ to eliminate linear dependencies. Then delete the first row of $\mathbf{D}'$ (which is $\mathbf{0}$ since we introduced a linear dependency).
5. Output the dual of the resulting matrix.

Also, as for perfect hints the parameters of the new lattice Λ' can be predicted: the dimension is unchanged, and the volume increases by a factor k under a primitivity condition, which is proved by Lemma 13.

4.3 Approximate Hints (conditioning)

Definition 26 (Approximate hint). *An approximate hint on the secret* $\mathbf{s}$ *is the knowledge of* $\mathbf{v} \in \mathbb{Z}^d$ *and* $l \in \mathbb{Z}$*, such that*

$$\langle \mathbf{s},\ \mathbf{v} \rangle + e = l,$$

where e *models noise following a distribution* $N_1(0, \sigma_e^2)$*, independent of* $\mathbf{s}$.

One can expect such hints from:

- any noisy side channel information about a secret coefficient. This is the case of our study in Section 6.1.
- decryption failures. In Section 6.2, we show how this type of hint can represent the information gained by a decryption failure.

To include this knowledge in the DBDD instance, we must combine this knowledge with the prior knowledge on the solution $\mathbf{s}$ of the instance.

Integrating an approximate hint into a DBDD instance Let $\mathbf{v} \in \mathbb{Z}^d$ and $l \in \mathbb{Z}$ be such that $\langle \mathbf{s}, \mathbf{v} \rangle \approx l$. Note that the hint can also be written as

$$\langle \bar{\mathbf{s}},\ \bar{\mathbf{v}} \rangle + e = 0 \tag{19}$$

where $\bar{\mathbf{s}}$ is the extended LWE secret as defined in Equation (6), $\bar{\mathbf{v}} := (\mathbf{v}\,;\, -l)$, and e has $N_1(0, \sigma_e^2)$ distribution. The unique shortest non-zero solution of $\mathsf{DBDD}_{\Lambda,\boldsymbol{\mu},\mathbf{\Sigma}}$, is also the unique solution of the instance $\mathsf{DBDD}_{\Lambda',\boldsymbol{\mu}',\mathbf{\Sigma}'}$ where

$$\Lambda' = \Lambda \tag{20}$$

$$\mathbf{\Sigma}' = \mathbf{\Sigma} - \frac{(\bar{\mathbf{v}}\mathbf{\Sigma})^T \bar{\mathbf{v}}\mathbf{\Sigma}}{\bar{\mathbf{v}}\mathbf{\Sigma}\bar{\mathbf{v}}^T + \sigma_e^2} \tag{21}$$

$$\boldsymbol{\mu}' = \boldsymbol{\mu} - \frac{\langle \bar{\mathbf{v}}, \boldsymbol{\mu} \rangle}{\bar{\mathbf{v}}\mathbf{\Sigma}\bar{\mathbf{v}}^T + \sigma_e^2}\bar{\mathbf{v}}\mathbf{\Sigma} \tag{22}$$

We note that Equation (20) come from

$$\Lambda' := \Lambda \cap \left\{ \mathbf{x} \in \mathbb{Z}^d \mid \langle \mathbf{x}, \bar{\mathbf{v}} \rangle + e = 0,\ \text{for all possible } e \sim N_1(0, \sigma_e^2) \right\} = \Lambda.$$

The new covariance and mean follow from Corollary 7.

Consistency with Perfect Hint Note that if $\sigma_e = 0$, we fall back to a perfect hint $\langle \mathbf{s}, \mathbf{v} \rangle = l$. The above computation of $\mathbf{\Sigma}'$ (21) (resp. $\boldsymbol{\mu}'$ (22)) is indeed equivalent to Equation (11) (resp. Equation (12)) from Section 4.1. Note however, in our implementation, that to avoid singularities, we require the span of $\mathrm{Span}(\mathbf{\Sigma} + \boldsymbol{\mu}^T \boldsymbol{\mu}) = \mathrm{Span}(\Lambda)$ (See the requirement in Equation (3)): If $\sigma_e = 0$, one *must* instead use a Perfect hint.

Multi-dimensional approximate hints The formulae of [27] are even more general, and one could consider a multidimensional hint of the form $\mathbf{sV} + \mathbf{e} = \mathbf{l}$, where $\mathbf{V} \in \mathbb{R}^{n \times k}$ and $\mathbf{e}$ a gaussian noise of any covariance $\mathbf{\Sigma_e}$. However, those general formulae require explicit matrix inversion which becomes impractical in large dimension. We therefore only implemented full-dimensional ($k = n$) hint integration in the *super-lightweight* version of our tool, which assumes all covariance matrices to be diagonal. These will be used for hints obtained from decryption failures in Section 6.2.

4.4 Approximate Hint (*a posteriori*)

In certain scenarios, one may more naturally obtain directly the a posteriori distribution of $\langle \mathbf{s}, \mathbf{v} \rangle$, rather than a hint $\langle \mathbf{s}, \mathbf{v} \rangle + e = l$ for some error e independent of $\mathbf{s}$. Such a scenario is typical in template attacks, as we exemplify via the single trace attack on Frodo from [11], which we study in Section 6.1.

Given the a posteriori distribution of $\langle \bar{\mathbf{s}}, \bar{\mathbf{v}} \rangle$, one can derive its mean μ_{ap} and variance σ_{ap}^2 and apply the corrections to compute the new mean and covariance exactly as in Equations (17) and (18).

$$\Lambda' = \Lambda \tag{23}$$

$$\boldsymbol{\mu}' = \boldsymbol{\mu} \cdot \mathbf{\Pi}_{\bar{\mathbf{v}}}^{\perp} + \mu_{\mathrm{ap}} \cdot \frac{\bar{\mathbf{v}}}{\|\bar{\mathbf{v}}\|^2} \tag{24}$$

$$\mathbf{\Sigma}' = (\mathbf{\Pi}_{\bar{\mathbf{v}}}^{\perp})^T \cdot \mathbf{\Sigma} \cdot \mathbf{\Pi}_{\bar{\mathbf{v}}}^{\perp} + \sigma_{\mathrm{ap}}^2 \cdot \frac{\bar{\mathbf{v}}^T \cdot \bar{\mathbf{v}}}{\|\bar{\mathbf{v}}\|^4} \tag{25}$$

4.5 Short vector hints

Definition 27 (Short vector hint). *A short vector hint on the lattice Λ is the knowledge of a short vector $\mathbf{v}$ such that*

$$\mathbf{v} \in \Lambda.$$

Note that such hints are not related to the secret, and are not expected to be obtained by side-channel information, but rather by the very design of the scheme. In particular, the lattice Λ underlying LWE instance modulo q contains the so-called q-vectors, i.e. the vectors $(q, 0, 0, \ldots, 0)$ and its permutations. These vectors are in fact implicitly exploited in the literature on the cryptanalysis of LWE since at least [26]. Indeed, in some regimes, the best attacks are obtained by 'forgetting' certain LWE equations, which can be geometrically interpreted as a projection orthogonally to that vector. Note that, among all hints, the short vector hints should be the last to be integrated. In our context, we need to generalize this idea beyond q-vector because the q-vectors may simply disappear after the integration of a perfect or modular hint. For example, after the integration of a perfect hint $\langle \mathbf{s}, (1, 1, \ldots, 1) \rangle = 0$, all the q-vectors are no longer in the lattice, but $(q, -q, 0, \ldots, 0)$ still is, and so are all its permutations.

Resolving the DBDD problem resulting from this projection will not directly lead to the original secret, as projection is not injective. However, as long as we keep $n + 1$ dimensions out of the $n + m + 1$ dimensions of the original LWE instance, we can still efficiently reconstruct the full LWE secret by solving a linear system over the rationals.

Integrating a short vector hint into a DBDD instance It is the case when the secret vector is short enough to be a solution after applying projection $\mathbf{\Pi}_{\mathbf{v}}^{\perp}$ on $\mathsf{DBDD}_{\Lambda,\mathbf{\Sigma},\boldsymbol{\mu}}$.

$$\Lambda' = \Lambda \cdot \mathbf{\Pi}_{\mathbf{v}}^{\perp} \tag{26}$$

$$\mathbf{\Sigma}' = (\mathbf{\Pi}_{\mathbf{v}}^{\perp})^T \cdot \mathbf{\Sigma} \cdot \mathbf{\Pi}_{\mathbf{v}}^{\perp} \tag{27}$$

$$\boldsymbol{\mu}' = \boldsymbol{\mu} \cdot \mathbf{\Pi}_{\mathbf{v}}^{\perp} \tag{28}$$

To compute a basis of Λ' one can simply apply the projection to all the vectors of its current basis, and then eliminate linear dependencies in the resulting basis using LLL.

Remark 28. Once a short vector hint $\mathbf{v} \in \Lambda$ has been integrated, Λ has been transformed into Λ'. And, if one has to perform another short vector hint integration $\mathbf{v}_1 \in \Lambda$, $\mathbf{v}_1$ should be projected onto Λ' with $\mathbf{v} \cdot \mathbf{\Pi}_{\Lambda'} \in \Lambda'$. In our implementation however, this has been taken into account and one can simply apply the same transformation as above, replacing a single vector $\mathbf{v}$ by a matrix $\mathbf{V}$.

The dimension of the lattice decreases by one (or by k, if one directly integrates a matrix of k vectors) and the volume of the lattice also decreases according to Fact 14. One can also predict the decrease of the determinant of $\mathbf{\Sigma}$ via the identity:

$$\mathrm{rdet}(\mathbf{\Sigma}') = \mathrm{rdet}(\mathbf{\Sigma}) \cdot \frac{\|\mathbf{v}\|^2}{\mathbf{v}\mathbf{\Sigma}\mathbf{v}^T}, \quad \text{or} \ \ \mathrm{rdet}(\mathbf{\Sigma}') = \mathrm{rdet}(\mathbf{\Sigma}) \cdot \frac{\det(\mathbf{V}\mathbf{V}^T)}{\det(\mathbf{V}\mathbf{\Sigma}\mathbf{V}^T)}. \tag{29}$$

Worthiness and choice of short vector hints Integrating such a hint induces a trade-off between the dimension and the volume, and therefore it is not always advantageous to integrate.

This raises the following potentially hard problem: given a set $\mathbf{W}$ of short vectors of Λ (viewed as a matrix), which subset $\mathbf{V} \subset \mathbf{W}$ of size k lead to the easiest DBDD instance? Because the hardness of the new problem grows with

$$\frac{\mathrm{rdet}(\mathbf{\Sigma}')}{\mathrm{Vol}(\Lambda')^2} = \frac{\mathrm{rdet}(\mathbf{\Sigma})}{\mathrm{Vol}(\Lambda)^2} \cdot \frac{\det(\mathbf{V}\mathbf{V}^T)^2}{\det(\mathbf{V}\mathbf{\Sigma}\mathbf{V}^T)} \tag{30}$$

In the case of an un-hinted DBDD instance directly obtained from the LWE problem, for $\mathbf{V}$ being the set of (primitive) q-vectors, the problem is easier: all subsets of size k lead to instances with the same parameters.

But this is not true anymore as soon as $\mathbf{\Sigma}$ has been altered or if the set $\mathbf{W}$ is arbitrary. For example, setting $\mathbf{\Sigma} = \mathbf{I}$, one simply wishes to minimize $\det(\mathbf{V}\mathbf{V}^T)$; but for an arbitrary set $\mathbf{W}$ the problem of finding the optimal subset $\mathbf{V} \subset \mathbf{W}$ is NP-hard [25], and remains NP-hard up to exponential approximation factors.

A natural approach to try to get an approximate solution in polynomial time consists in making sequential greedy choices. This involves computing $|\mathbf{V}|\cdot|\mathbf{W}|$ many matrix-vector products over increasingly large rationals, and appeared painfully slow in practice for making prediction on cryptographically large instances. Fortunately, in the typical cases where the vectors of $\mathbf{W}$ are the q-vectors, this can be made somewhat practical (See Section 6.3 for example).

Remark 29. When the basis of an LWE-lattice is given in its systematic form, the q-vectors are already explicitly given to lattice reduction algorithms, and these algorithms will implicitly make use of them when they are worthy, as if we had integrated them. The reason is that lattice reduction algorithm naturally work with projected sublattices, and if a q-vector is shorter than what the algorithm can produce, those q-vectors will remain untouched at the beginning of the basis; the reduction algorithm will effectively work on the lattice projected orthogonally to them. In other words, integrating q-vectors is important to understand and predict how lattice reduction algorithm will work, but, in certain cases they may be automatically detected and exploited by lattice reduction algorithms themselves.

5 Implementation

5.1 Our sage implementation

We propose three implementations of our framework, all following the same python/sage 9.0 API.[9] More specifically, the API and some common functions are defined in DBDD_generic.sage, as a class `DBDD_Generic`. Three derived classes are then given:

1. The class `DBDD` (provided in DBDD.sage) is the *full-fledged* implementation: i.e. it fully maintains all information about a DBDD instance as one integrates hints: the lattice Λ, the covariance matrix $\mathbf{\Sigma}$ and the average μ. While polynomial time, maintaining the lattice information can be quite slow, especially since consecutive intersections with hyperplanes can lead to manipulations on rationals with large denominators. It also allows to finalize the attack, running the homogenization, isotropization and lattice reduction, based on the fplll library available through sage.
   We note that if one were to repeatedly use perfect or modular hints, a lot of effort would be spent on uselessly alternating between the primal and the dual lattice. Instead, we implement a caching mechanism for the primal and dual basis, and only update them when necessary.
2. The class `DBDD_predict` (provided in DBDD_predict.sage) is the *lightweight* implementation: it only fully maintains the covariance information, and the parameters of the lattice (dimension, volume). It must therefore work under assumptions about the primitivity of the vector $\mathbf{v}$; in particular, it cannot detect hints that are redundant. If one must resort to this faster variant on large instances, it is advised to consider potential (even partial) redundancy between the given hints, and to run a comparison with the previous on small instances with similarly generated hints.
3. The class `DBDD_predict_diag` (provided in DBDD_predict_diag.sage) is the *super-lightweight* implementation. It maintains the same information as the above, but requires the covariance matrix to remain diagonal at all times. In particular, one can only integrate hints for which the directional vector $\mathbf{v}$ is colinear with a canonical vector.

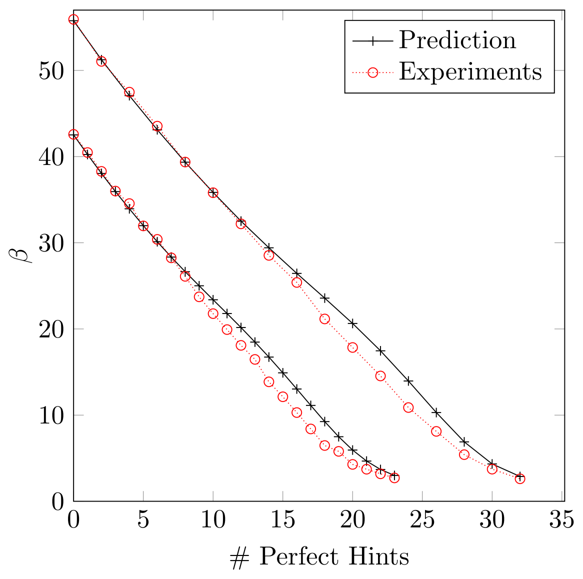
5.2 Tests and validation

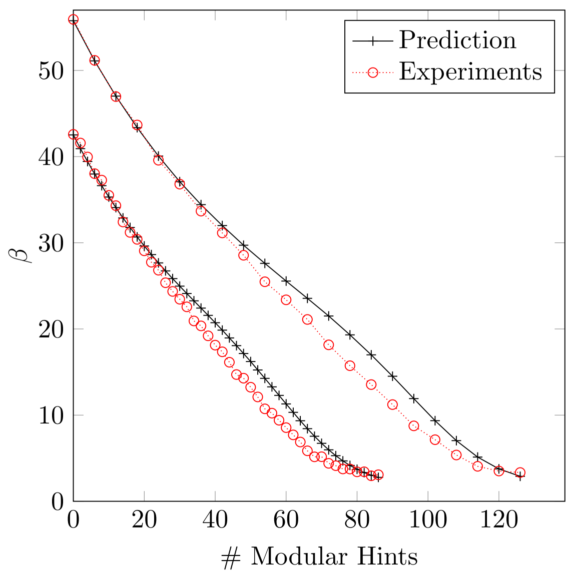
In Appendix A, we present a demonstration of our tool with some extracts of Sage 9.0 code. We implement two tests to verify the correctness of our scripts, and more generally the validity of our predictions.

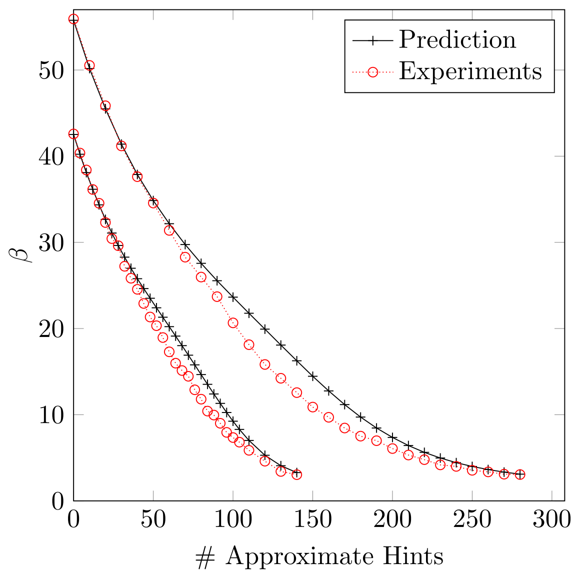
Consistency checks. Our first test (check_consistency.sage) simply verifies that all three classes always agree perfectly. More specifically we run all three versions on a given instances, integrating the same random hint in all of them, and compare their hardness prediction. We first test using the full-fledged version that the primitivity condition does hold, and discard the hint if not, as we know that predictions cannot be correct on such hints. This verification passes.

Prediction verifications. We now verify experimentally the prediction made by our tool for various types of hints, by comparing those predictions to actual attack experiments (see compare_usvp_models.sage for the prediction without hints and prediction_verifications.sage for the prediction with hints). This is done for a given set of LWE parameters, and increasing the number of hints. The details of the experiments and the results are given in Figure 5.

[9] While we would have preferred a full python implementation, we are making a heavy use of linear algebra over the rationals for which we could find no convenient python library.







LWE Parameters	
$n = m = 70, \quad q = 3301, \quad \sigma = 20$	
$n = m = 80, \quad q = 3301, \quad \sigma = 20$	
Hint Type	Parameter
Perfect hints	–
Modular hints	mod 11
Approximate hints	$\sigma_\epsilon^2 = 3$

The hint vectors $\mathbf{v}$ were chosen as random ternary vectors of weight 5.

Fig. 5. Experimental verification of the security decay predictions for each type of hints. Each data point was averaged over 256 samples.

While our predictions seem overall accurate, we still note a minor discrepancy of up to 2 or 3 bikz in the low blocksize regime. This exceeds the error made by prediction on the attack without any hint, which was below 1 bikz, even in the same low blocksize regime. We suspect that this discrepancy is due to residual q-vectors, or small combinations of them, that are hard to predict for randomly generated hints, but would still benefit by lattice reduction. We tested that hypothesis by running similar experiments, but leaving certain coordinates untouched by hints, so to still explicitly know some q-vectors for short-vector hint integration, if they are "worthy". This didn't to improve the accuracy of our prediction. We are at the moment unable to explain it. We nevertheless find our predictions satisfactory, considering that even without considering hints, previous predictions [4] were much less accurate (see Figure 4).

6 Applications examples

6.1 Hints from side channels: application to [11]

In [11], W. Bos et al. study the feasibility of a single trace power analysis of the Frodo Key Encapsulation Mechanism [32]. Specifically, in a first approach, they analyze the possibility of a divide-and-conquer template attack targeting two points of interest (the loading of the secret and the $\mathbb{Z}_q$ multiplication). The latter targets each coefficient of the secret $\mathbf{s}_i$ separately. They designed a distinguisher for the possible values that each secret coefficient can have. The side-channel simulator was provided to us by the authors of [11]. [10] and we were able to re-generate all their data using Matlab.

In more details, for Frodo, the secret coefficients can take several values in a set that we denote L. For example, with NIST1 parameters, $L = \{-11, \cdots, 11\}$. The distinguisher's strategy consists of associating a score to these potential values, that we denote here $(S[x])_{x \in L}$. Consider a secret coefficient $\mathbf{s}_i$, its *best guess* is defined as the possible value $g \in L$ that has the maximum score, *i.e.* $S[g] = \max_{x \in L}(S[x])$. Let S be a score, we define $\mathsf{bestguess}(S) \in L$ as follows;

$$\mathsf{bestguess}(S) := \operatorname{argmax}_{x \in L}(S[x]). \qquad \text{(Best guess)}$$

We refer to [11] for more information about the technique that is employed to derive the score from simulated side-channels. It is actually parametrized by the standard deviation of the side-channel noise. The less side-channel noise there is, the more accurate the best guess is. When there is no noise, the best guess is always correct. For our proof of concept, the data was generated for a simulated noise having a standard deviation of 0.0045. The latter is considered as a realistic modelization (see the vertical line in [11, Fig. 2b]).

As an example, using NIST1 parameters, we derived several examples of measured score $(S[-11], \cdots, S[11])$ corresponding to several secret coefficients in Table 1. The first line corresponds to a secret equal to 0, the second line to 1 and the third and fourth line to -1. The last line is an example of failed guessing because we see that $\mathsf{bestguess}(S) \neq -1$. We remark that the values having the opposite sign are assigned a very low score, we conjecture that it is because the sign is filling the register and then the Hamming weight of the register will be very far from the correct one.

With this template attack, Bos et al. could not conclude the attack with a key recovery even though much information leaked about the secret. Frustratingly, a bruteforce attack cannot lead to any security threat as stated in [11, Section 3]. They actually pointed out an interesting open question: "possibly [...] novel lattice reduction algorithms [can] take into account side-channel information". The following solves this open question by combining the knowledge obtained in

[10] We are very thankful to Bos et al. for sharing their source code.

Table 1. Examples of scores associated to the secret values $\mathbf{s}_i \in \{0, \pm 1\}$, after the side-channel analysis of [11] for NIST1 parameters. The best score in each score table is highlighted. This best guess is correct for the first 3 score table, but incorrect for the last one.

$\mathbf{s}_i$	S																						
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
0	-4098	-3918	-4344	-2580	-3212	-3108	-3758	-3155	-3583	-3498	-3900	**-340**	-380	-367	-452	-818	-975	-933	-1084	-368	-459	-453	-592
1	-3273	-3114	-3491	-1951	-2495	-2405	-2972	-2445	-2819	-2744	-3098	-365	**-325**	-328	-338	-546	-657	-627	-737	-333	-344	-342	-407
-1	-341	-335	-352	-465	-358	-369	-329	-362	-331	-334	**-328**	-3712	-3079	-3195	-2656	-1696	-1461	-1521	-1329	-3231	-2648	-2685	-2201
-1	-306	-298	-319	-414	-314	-323	**-290**	-317	-291	-293	-291	-3608	-2982	-3097	-2564	-1617	-1385	-1444	-1256	-3132	-2556	-2593	-2115

the divide-and-conquer template attack of [11] with our framework. We introduce a two-phase attack: First, we make an offline phase that consists in deriving aposteriori distributions for several sets of parameters of Frodo (CCS1, CCS2, CCS3, CCS4, NIST1 and NIST2) from the experimental data of [11]. Secondly, we attack a specific instance using the obtained best guesses and the a posteriori distribution derived in phase one.

First phase. We first learn a posteriori distributions from the experimental data of [11]. The score table for a secret coefficient $\mathbf{s}_i$ is denoted S_i. We can then derive the a posteriori probability distribution of the secret *knowing the value of the best guess.* Let $x, g \in L$ and $0 \leq i \leq n-1$,

$$\mathcal{D}_{i,g}^{\text{apost}}(x) = P[\mathbf{s}_i = x | \ \mathsf{bestguess}(S_i) = g],$$

and more precisely its center denoted $\mu_{i,g}^{\text{apost}}$ and its variance denoted $v_{i,g}^{\text{apost}}$.

Similarly as the authors of [11], we make an independence assumption. We assume that the obtained score only depends on the secret coefficient $x \in L$ and the obtained best guess $g \in L$. We then omit the index i and denote for $x, g \in L$,

$$\mathcal{D}_{g}^{\text{apost}}(x) := P[\mathbf{s}_0 = x | \ \mathsf{bestguess}(S_0) = g],$$

with μ_g^{apost} its center and v_g^{apost} its variance.

We experimentally computed the latter distribution using a large amount of data obtained from the Matlab code of [11]: we considered $80 \times n$ secret coefficients in $\mathbb{F}_q$ with their associated score table S, which corresponds to $\approx 50,000$ pairs (secret coefficient s_i, score table S) for NIST1 parameters. More specifically, it consists in putting the secret coefficients in different buckets depending on the value of the best guess $g = \mathsf{bestguess}(S)$ given by their score table S. And, for each bucket, we count the occurrences of each secret value and deduce the center μ_g^{apost} and variance v_g^{apost}. In Table 2, we provide several characteristics of $\mathcal{D}_g^{\text{apost}}$ for the the NIST1 parameters. We noticed several details about the experimentally derived a posteriori distribution.

- On all sets of parameters, when the best guess is 0, the secret has overwhelming probability to be 0 as well. This is not surprising as it is the only case where the Hamming weight of the secret is zero.

Best guess g ...	-4	-3	-2	-1	0	1	2	3	4	...
μ_g^{apost}	-4.58	-2.66	-2.42	-1.52	0.00	1.32	1.76	3.07	4.09	
v_g^{apost}	1.09	0.837	0.435	1.99	0.00	0.399	0.765	0.444	0.159	
$\mathcal{D}_g^{\text{apost}}(g)$	0.73	0.46	0.63	0.87	1.0	0.71	0.66	0.99	0.95	

Table 2. A Posteriori Distribution for NIST1 parameters

	NIST1	NIST2	CCS1	CCS2	CCS3	CCS4
Attack without hints (bikz)	487	708	239	448	492	584
Attack with hints (bikz)	**337**	**471**	**190**	**297**	**351**	**294**
Attack with hints & guesses (bikz)	**298**	**403**	**126**	**110**	**209**	**206**
Number of guesses k	50	100	100	350	300	150
Success probability	0.51	0.58	0.61	0.66	0.62	0.55

Table 3. Cost of the attacks without/with hints without/with guesses.

– The values are derived from $80 \times n$ coefficients following the secret key distribution. Then, the elements of the tail of the distribution are rarely sampled. So, the learning set provides a small amount of samples to compute $\mu_g^{\text{apost}}, v_g^{\text{apost}}$ and $\mathcal{D}_g^{\text{apost}}(g)$ for tail values. These tail values (for NIST1 parameters we consider $\{-11, \ldots, -7, 7, \ldots, 11\}$ as tail values) are sampled very rarely so we ignore the hints involving them due to the lack of learning data.

Second phase. We first instantiate a DBDD instance with a chosen set of parameters. Then we assume that each secret coefficient has been associated a best guess[11]. We thus introduce n (a posteriori) *approximate hints* to our DBDD instance. Let $0 \leq i \leq n-1$, $\mathbf{v}_i$ be the i-th canonical vector and g_i be the best guess associated to the i-th secret coefficient. The approximate hint is

$$\langle \mathbf{s},\ \mathbf{v}_i \rangle = \mu_{g_i}^{\text{apost}} + e \tag{31}$$

where e models the noise following a distribution $N_1(0, v_{g_i}^{\text{apost}})$ where $\mu_{g_i}^{\text{apost}}$ and $v_{g_i}^{\text{apost}}$ have been computed in phase one. One can remark that according to Table 2, $g = 0$ has variance 0.000 so the approximate hint is actually a perfect hint. Once all these hints are included, one can derive the corresponding security against lattice reduction of the DBDD instance with the super lightweight version of our tool.

Results. One can reproduce this attack with detailed computations using the Sage 9.0 script exploiting_SCA_from_Bos_et_al.sage. The experimentally derived data is in the folder Frodo_Single_data for which we recall that it was generated with a simulated noise variance of 0.0045. We note that the obtained security fluctuates a bit from instance to instance, as it depends on the strength of the hints, which themselves depend on the randomness of the scheme and of the leakage noise. In the first two lines of Table 3, we show the new security with the inclusion of the approximate hints averaged on 100 tests per sets of parameters.

Guessing. To improve the attack further, one can note from Table 2 that certain best guesses have a very high probability of being correct, and assuming it is, one can replace an approximate

[11] The associated best guess is not directly simulated with the Matlab tool for compatibility reasons with Sage 9.0. For each secret coefficient $\mathbf{s}_i$, we generate a best guess g_i exactly as the attack of [11] would do. We deduce the distribution $P[\mathsf{bestguess}(S_i) = g \mid \mathbf{s}_i = x]$ from the tables S of the large amount of data. And, we use the latter distribution to generate the best guesses.

hint with a perfect one. More specifically, let us suppose that a coefficient $\mathbf{s}_i$ is associated a best guess g_i. If the probability that the best guess is correct (denoted before as $\mathcal{D}_{g_i}^{\text{apost}}(g_i)$) is large, choosing g_i for the secret coefficient has good chances to be valid. For example, in Table 2, the coefficients that are best candidates for guessing are the ones that are associated to the best guesses $g_i = 3$, then 4, then -1 and so on (note that the coordinates with best-guess 0 have already been integrated as perfect hints in the first phase). This hybrid attack exploiting hints, guesses and lattice reduction, works as follows:

1. Include all the approximate and perfect hints from Equation (31),
2. Order the coefficients of the secret $\mathbf{s}_i$ according to $\mathcal{D}_{g_i}^{\text{apost}}(g_i)$ where g_i is their associated best guesses,
3. Include the following perfect hints for the k first coefficients: $\langle \mathbf{s},\ \mathbf{v}_i\rangle = g_i$ where g_i are the best guesses and then solve and check the solution.

Increasing the number of guesses k leads to a trade-off between the cost of the attack and its success probability. We here chose a success probability of about 1/2, while decreasing the cost of the attack by 39 to 187 bikz depending on the parameter set. Given that 1 bits of security corresponds roughly to 3 or 4 bikz, this is undoubtedly advantageous. More specifically, this hybrid technique works very well with the CCS parameters but is less spectacular for NIST parameters as the success probability decreases faster (see Table 3).

Second-guessing. It should be noted that, given a single trace, one cannot naively retry the attack to boost its success probability. Indeed, the "second-best" guess may already have a much lower success probability than the first, and the problem boils down to setting up a hybrid attack mixing lattice reduction within our framework, and key-ranking. We avoided such consideration by restricting ourselves to rather high success probability.

6.2 Hints from decryption failures: application to [16]

Another kind of hint our framework can model are hints provided by decryption failures. For a single-bit LWE encryption scheme, a decryption failure occurs when the random short vector $\mathbf{w}$ used during encryption is such that $|\langle \mathbf{s}, \mathbf{w}\rangle| \geq t$ for some t, typically $t = q/4$.

In fact, we can even assume to know the "side" of the decryption failure, i.e. we can assume we know that $\langle \mathbf{s}, \mathbf{w}\rangle \geq t$. Indeed, this can be guessed with probability 1/2 for the first failure, and it can be deduced for subsequent failures using the fact that those sides are strongly correlated (see Section 4.3 in [16] for example). For multi-bit encryption, using either ring-element or matrices for secrets, similar techniques allow to "locate" the failure, and therefore obtain information of this form.

We will here consider the case of the Chosen-Ciphertext-Attack (CCA) secure variant of such schemes, typically obtained by variants of the Fujikasi-Okamoto transform. In this case, the attacker does not control the short vector $\mathbf{w}$, as it is generated following the randomness of a hash function.

Following our framework, it would be tempting to simply construct the conditional distribution of $\langle \mathbf{s}, \mathbf{w}\rangle$ given that $|\langle \mathbf{s}, \mathbf{w}\rangle| \geq t$, and integrate this as an a posteriori hint with $\mathbf{v} = \mathbf{w}$. However, this modeling would actually lose a lot of information. Indeed, such hints are designed in the case where one first chooses $\mathbf{w}$ independently of $\mathbf{s}$, and then learns partial information on $\langle \mathbf{s}, \mathbf{w}\rangle$. The setting here is quite different: one instead samples $\mathbf{w}$ following a prescribed distribution, until failure occurs. In other word, $\mathbf{w}$ is sampled on a prescribed distribution, and conditioned on $\langle \mathbf{s}, \mathbf{w}\rangle \geq t$. In particular it is not sampled independently of the secret $\mathbf{s}$, and it carries information on $\mathbf{s}$ in all directions.

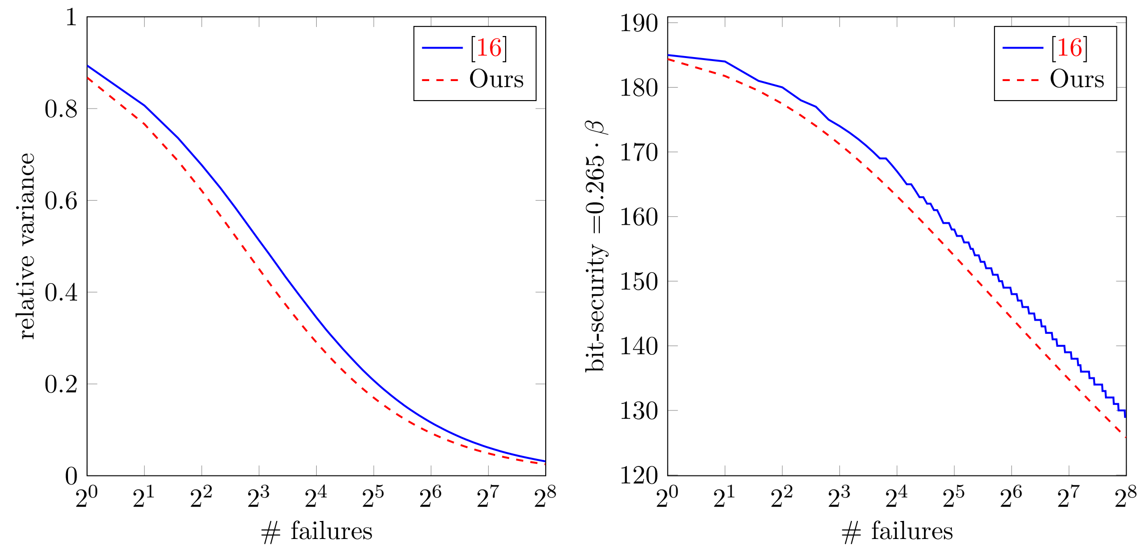


Fig. 6. Security decrease as a function of the number of failure in FRODOKEM-976.

For the sake of simplicity, let us assume that the norm of $\mathbf{s}$ is exactly $\ell = \sqrt{n}\sigma$; making such a guess is rather inconsequential given how concentrated the norm of a high dimensional Gaussian is. Let us assume that $\mathbf{w}$ also follows a Gaussian of covariance $\tau^2\mathbf{I}$, before imposing the condition. After conditioning, $\mathbf{w}$ decomposes as $\mathbf{w} = \alpha\mathbf{s}/\ell + \mathbf{w}'$, where $\mathbf{w}'$ is a Gaussian of covariance $\tau^2\mathbf{\Pi}_{\mathbf{s}}^{\perp}$, and α is independent of $\mathbf{w}'$ and follows a distribution that we denote $G_\tau^{\geq t/\ell}$, the unidimensional Gaussian of variance τ^2 conditioned on $\alpha \geq t/\ell$. One can check that the $\mathbb{E}_{X \leftarrow G_\tau^{\geq t/\ell}}[(t/\ell - X)^2] \leq \tau^2$ for any $t/\ell \geq 0$. This means that we can write $\mathbf{w} = t/\ell^2 \cdot \mathbf{s} + \mathbf{e}$ for some error $\mathbf{e}$ of (ill-centered) covariance $\mathbf{\Sigma_e} \leq \tau\mathbf{I}$.

Rewriting the above equality, we finally obtain a full dimensional approximate hint of the form

$$\mathbf{s} = \frac{\ell^2}{t}\mathbf{w} + \mathbf{e}'$$

with an error $\mathbf{e}' = -\frac{\ell^2}{t}\mathbf{e}$ of (uncentered) covariance $\tau^2\ell^4/t^2 \cdot \mathbf{I}$.

We can now compare the results of our prediction to prior work that used several other methodologies such as [16,21,17,15]. We choose to compare with [16] on FRODOKEM-976, for which the data can be reproduced[12], and for which $\mathbf{w}$ is indeed very close to Gaussian. We note that both methods use different simplications or heuristics, nevertheless they produce essentially similar predictions, as shown in Figure 6. The data using our framework has been acquired with the script exploiting_decryption_failures.sage.

Furthermore, one could try to refine the estimate of the average and variance of $\mathbf{e}$, which can improve in direction of $\mathbf{w}$. However, this would force us to deal with non-diagonal covariance matrices, which generically significantly slows down our script, and would require further optimizations to be doable in practice. The exploration of such improvements is left as future work.

Table 4. New security estimates

	LAC-128	LAC-192	LAC-256
Attack without hint (bikz)	509.03	985.64	1104.83
Attack with hint (bikz)	505.94	982.74	1101.61

	R5ND_{1}KEM_0d	R5ND_{3}KEM_0d	R5ND_{5}KEM_0d
Attack without hint (bikz)	494.39	658.67	877.71
Attack with hint (bikz)	493.36	657.58	876.62

	ntruhps2048509	ntruhps2048677	ntruhps4096821
Attack without hint (bikz)	373.46	516.23	618.58
Attack with hint (bikz)	372.43	515.23	617.57

6.3 Hints from structural design

LAC is a Ring-LWE round two candidate of the NIST post-quantum competition [28]. The secrets are two polynomials $\mathbf{s}_0, \mathbf{s}_1$ (denoted $\mathbf{s}$ and $\mathbf{e}$ in the specifications) whose coefficients follow a distribution $\psi^{n,h}$, the uniform distribution over ternary vectors $\{-1, 0, 1\}^n$ with exactly $h/2$ ones and $h/2$ minus ones. Thus, two structural perfect hints can be derived:

$$\sum_{i=0}^{n-1} \mathbf{s}_0[i] = 0 \text{ and } \sum_{i=0}^{n-1} \mathbf{s}_1[i] = 0.$$

The same structure appears in the submissions Round5 (only for $\mathbf{s}_0$) and NTRU-HPS as they also require the number of -1 coefficients to be balanced with number of 1 coefficients of their ternary polynomial. This new knowledge has been included in the security analysis and the results are stored in Table 4. One can check the experiments by running the scripts exploiting_design_LAC.sage, exploiting_design_round5.sage and exploiting_design_ntru.sage. For Round5, we arbitrarily chose for our testing the parameter set R5ND_$\{1, 3, 5\}$KEM_0d.

Remark 30. Note, however, that integrating such hints removes some q-vectors from the lattice. For LAC and NTRU-HPS, we note that while q-vectors are not in the lattice, a difference of 2 such vectors is still in it, for example the short vector hint $(q, -q, 0, 0, \ldots, 0) \in \Lambda$. We iteratively integrate $(q, -q, 0, 0, \ldots, 0)$, $(0, q, -q, 0, \ldots, 0)$, $(0, 0, q, -q, \ldots, 0)$, $\ldots$ until such hints are not worthy anymore, i.e. until such hints do not decrease the cost of the attack anymore.

Remark 31. A similar structure is present in the candidate NTRU-Prime in its streamlined and LPR versions [8]. In the secret vector, the number of ± 1's is fixed to an integer w without knowing the exact number of positive and negative ones. Thus, one can include a modular hint

$$\sum_{i=0}^{n-1} \mathbf{s}_0[i] = w \mod 2.$$

The loss of security is however essentially negligible.

[12] https://github.com/KULeuven-COSIC/PQCRYPTO-decryption-failures/

References

1. M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the LWE, NTRU schemes! In *International Conference on Security and Cryptography for Networks*, pages 351–367. Springer, 2018.
2. M. R. Albrecht, A. Deo, and K. G. Paterson. Cold boot attacks on ring and module LWE keys under the NTT. In *IACR TCHES*, volume 2018, pages 173–213, Aug. 2018.
3. M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens. The general sieve kernel and new records in lattice reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 717–746. Springer, 2019.
4. M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 297–322. Springer, 2017.
5. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange—a new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, 2016.
6. S. Bai, S. Miller, and W. Wen. A refined analysis of the cost for solving LWE via uSVP. Cryptology ePrint Archive, Report 2019/502, 2019.
7. S. Bai, D. Stehlé, and W. Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 369–404. Springer, 2018.
8. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. PQC Round-2 candidate: NTRU Prime. Technical report, NIST, 2019. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
9. M. Bolboceanu, Z. Brakerski, R. Perlman, and D. Sharma. Order-lwe and the hardness of ring-lwe with entropic secrets. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 91–120, 2019.
10. J. Bootle, C. Delaplace, T. Espitau, P.-A. Fouque, and M. Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 494–524. Springer, 2018.
11. J. W. Bos, S. Friedberger, M. Martinoli, E. Oswald, and M. Stam. Assessing the feasibility of single trace power analysis of frodo. In *SAC*, 2018.
12. L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. Flush, gauss, and reload–a cache attack on the bliss lattice-based signature scheme. In *IACR TCHES*, pages 323–345. Springer, 2016.
13. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833, pages 1–20, Dec. 2011.
14. J. H. Cheon, D. Kim, J. Lee, and Y. Song. Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR. In *International Conference on Security and Cryptography for Networks*, pages 160–177. Springer, 2018.
15. J.-P. D'Anvers, M. Rossi, and F. Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. Cryptology ePrint Archive, Report 2019/1399, 2019.
16. J.-P. D'Anvers, F. Vercauteren, and I. Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. *IACR Cryptology ePrint Archive*, 2018:1089, 2018.
17. J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In *IACR International Workshop on Public Key Cryptography*, pages 565–598. Springer, 2019.
18. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–278. Springer, 2010.
19. O. Garcia-Morchon, Z. Zhang, S. Bhattacharya, R. Rietman, L. Tolhuizen, J.-L. Torre-Arce, H. Baan, M.-J. O. Saarinen, S. Fluhrer, T. Laarhoven, and R. Player. Round5. Technical report, NIST, 2019.
20. L. Groot Bruinderink and P. Pessl. Differential fault attacks on deterministic lattice signatures. In *IACR TCHES*, volume 2018, pages 21–43, Aug. 2018.
21. Q. Guo, T. Johansson, and A. Nilsson. A generic attack on lattice-based schemes using decryption errors. Cryptology ePrint Archive, Report 2019/043, 2019.
22. J. Hoffstein, N. Howgrave-Graham, J. Pipher, and W. Whyte. Practical lattice-based cryptography: NTRU-Encrypt and NTRUSign. In *The LLL Algorithm*, pages 349–390. Springer, 2009.
23. N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In A. Menezes, editor, *CRYPTO 2007*, volume 4622, pages 150–169, Aug. 2007.
24. R. Kannan. Minkowski's convex body theorem and integer programming. In *Mathematics of operations research*, volume 12, pages 415–440. INFORMS, 1987.
25. L. Khachiyan. On the complexity of approximating extremal determinants in matrices. volume 11, pages 138–153. Elsevier, 1995.

26. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, pages 319–339. Springer, 2011.
27. L.-P. Liu. Linear transformation of multivariate normal distribution: Marginal, joint and posterior, Accessed on September 2019. http://www.cs.columbia.edu/~liulp/pdf/linear_normal_dist.pdf.
28. X. Lu, Y. Liu, D. Jia, H. Xue, J. He, Z. Zhang, Z. Liu, H. Yang, B. Li, and K. Wang. PQC Round-2 candidate: LAC. Technical report, NIST, 2019. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
29. J. Martinet. *Perfect lattices in Euclidean spaces*, volume 327. Springer, 2013.
30. D. Micciancio. Lecture notes of CSE206A: Lattices algorithms and applications (Spring 2014)-Duality, Accessed on September 2019. http://cseweb.ucsd.edu/classes/sp14/cse206A-a/lec3.pdf.
31. D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. volume 37, pages 267–302. SIAM, 2007.
32. M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM. Technical report, NIST, 2019.
33. P. Nguyen. Giophanthus and *LWR-based submissions, 2019. Comment on the NIST PQC forum, https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/nZBIBvYmmUI/J0pug16CBgAJ.
34. T. Poppelmann, E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, P. Schwabe, D. Stebila, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart. NewHope. Technical report, NIST, 2019.
35. P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin. Side-channel assisted existential forgery attack on Dilithium - A NIST PQC candidate. Cryptology ePrint Archive, Report 2018/821, 2018.
36. P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin. Exploiting determinism in lattice-based signatures: Practical fault attacks on pqm4 implementations of nist candidates. Asia CCS '19, page 427–440. Association for Computing Machinery, 2019.
37. P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehlé. CRYSTALS-KYBER. Technical report, NIST, 2019.
38. Y. Yu and L. Ducas. Second order statistical behavior of LLL and BKZ. In *International Conference on Selected Areas in Cryptography*, pages 3–22. Springer, 2017.
39. Z. Zhang, C. Chen, J. Hoffstein, W. Whyte, J. M. Schanck, A. Hulsing, J. Rijneveld, P. Schwabe, and O. Danba. PQC Round-2 candidate: NTRU. Technical report, NIST, 2019. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

A Demonstration of our tool

This Section is a demonstration of our tool. We will integrate many types of hints with several versions of the implementation. The pedagogic purpose of this explanation makes the scenarios very unrealistic in terms of side information leakages. For realistic uses, we refer to Section 6.

A.1 Demonstration of the full-fledged version

The full-fledged implementation is called when the class of the instance is DBDD. Let us create a small LWE instance and estimate its security in bikz (see Section 3.4).

```
sage: load("../framework/instance_gen.sage")
....: n = 70
....: m = n
....: q = 3301
....: D_s = build_centered_binomial_law(40)
....: D_e = D_s
....: A, b, dbdd = initialize_from_LWE_instance(DBDD, n, q, m, D_e, D_s)
....: # In such parameter range, no need to integrate q-vectors
....: beta, delta = dbdd.estimate_attack()
```

```
      Build DBDD from LWE
 n= 70   m= 70   q=3301
       Attack Estimation
 dim=141     δ=1.012362      β=45.40
```

Our full-fledged implementation contains an attack procedure that runs BKZ with iterating gradually the block size. It then compares the recovered secret with the actual one.

```
sage: t = cputime()
....: secret = dbdd.attack()
....: str(cputime(t)) + "CPU Seconds for the attack"
```

```
       Running the Attack
Running BKZ-42   Success !

120.73061799999999 CPU Seconds for the attack
```

Here, the block size stopped at 42 while an average blocksize of 45.40 has been estimated. Let us now create four vectors $\mathbf{v}$ for making perfect hints (See Section 4.1). To simulate side information, we compute the hints with the function dbdd.leak($\mathbf{v}$) that returns $l = \langle \mathbf{s},\ \mathbf{v} \rangle$.

```
sage: # Simulating perfect hints
....: v0 = vec([randint(0, 1) for i in range(m + n)])
....: v1 = vec([randint(0, 1) for i in range(m + n)])
....: v2 = vec([randint(0, 1) for i in range(m + n)])
....: v3 = vec([randint(0, 1) for i in range(m + n)])
....: # Computing l = <vi, s>
....: dbdd.leak(v0), dbdd.leak(v1), dbdd.leak(v2), dbdd.leak(v3)
```

```
(27, -62, -45, -47)
```

Let us now integrate the perfect hints into our instance.

```
sage: # Integrate perfect hints
....: _ = dbdd.integrate_perfect_hint(v0, 27)
....: _ = dbdd.integrate_perfect_hint(v1, -62)
....: _ = dbdd.integrate_perfect_hint(v2, -45)
....: _ = dbdd.integrate_perfect_hint(v3, -47)
```

```
 integrate perfect hint   u0 + u1 + u7 + u8 + u9 + ... = 27
 Worthy hint !   dim=140, δ=1.01252643, β=41.93
 integrate perfect hint   u0 + u2 + u8 + u9 + u10 + ... = -62
 Worthy hint !   dim=139, δ=1.01275412, β=38.42
 integrate perfect hint   u0 + u1 + u3 + u4 + u7 + ... = -45
 Worthy hint !   dim=138, δ=1.01293851, β=34.78
 integrate perfect hint   u1 + u9 + u11 + u12 + u13 + ... = -47
 Worthy hint !   dim=137, δ=1.01314954, β=30.91
```

The cost of the lattice attack has decreased by ≈ 14 bikz. Let us now create four vectors $\mathbf{v}$ for making modular hints (See Section 4.2). To simulate side information, we compute the hint with the function dbdd.leak($\mathbf{v}$) with different moduli. We then get $l = \langle \mathbf{s}, \ \mathbf{v} \rangle \bmod k$.

```
sage: # Simulating modular hints
....: v0 = vec([randint(0, 1) for i in range(m + n)])
....: v1 = vec([randint(0, 1) for i in range(m + n)])
....: v2 = vec([randint(0, 1) for i in range(m + n)])
....: v3 = vec([randint(0, 1) for i in range(m + n)])
....: # Computing l = <vi, s> mod k
....: dbdd.leak(v0)%2, dbdd.leak(v1)%3, dbdd.leak(v2)%4, dbdd.leak(v3)%5
```

```
(1, 1, 2, 3)
```

Let us now integrate the modular hints into our instance. We assume smoothness. In other words, the lattice is sparsified according to Equation (14) but the covariance matrix and average remain the same.

```
sage: # Integrate modular hints
....: _ = dbdd.integrate_modular_hint(v0, 1, 2, True)
....: _ = dbdd.integrate_modular_hint(v1, 1, 3, True)
....: _ = dbdd.integrate_modular_hint(v2, 2, 4, True)
....: _ = dbdd.integrate_modular_hint(v3, 3, 5, True)
```

```
integrate modular hint   (smooth)   u2 + u3 + u4 + ... = 1 MOD 2
        Worthy hint !   dim=137, δ=1.01318729, β=30.55
integrate modular hint   (smooth)   u0 + u2 + u10 + ... = 1 MOD 3
        Worthy hint !   dim=137, δ=1.01319931, β=29.98
integrate modular hint   (smooth)   u0 + u4 + u9 + ... = 2 MOD 4
        Worthy hint !   dim=137, δ=1.01327415, β=29.24
integrate modular hint   (smooth)   u2 + u3 + u6 + ... = 3 MOD 5
        Worthy hint !   dim=137, δ=1.01331577, β=28.37
```

As modular hints contain less information than perfect ones, especially for low modulus, the cost of the lattice attack decreased by only 3 bikz. Let us do the same for approximate hints (See Sections 4.3 and 4.4). To simulate side information, we compute the hint with the function dbdd.leak($\mathbf{v}$) and manually change the value to represent the measurement noise. We then get $l \approx \langle \mathbf{s}, \ \mathbf{v} \rangle$.

```
sage: # Simulating approximate hints
....: v0 = vec([randint(0, 1) for i in range(m + n)])
....: v1 = vec([randint(0, 1) for i in range(m + n)])
....: v2 = vec([randint(0, 1) for i in range(m + n)])
....: v3 = vec([randint(0, 1) for i in range(m + n)])
....: # Computing l = <vi, s> + noise
....: dbdd.leak(v0) + 2, dbdd.leak(v1) + 1, dbdd.leak(v2) - 1, dbdd.leak(v3)
```

```
(-19, -29, -16, 1)
```

Let us now integrate the approximate hints into our instance. We assume that we want to condition the new information with the prior one (See Section 4.3) and not to erase the previous distribution (See Section 4.4).

```
sage: # Integrate approximate hints
....: var = 10
....: _ = dbdd.integrate_approx_hint(v0, -19, var, aposteriori=False)
....: _ = dbdd.integrate_approx_hint(v1, -29, var, aposteriori=False)
....: _ = dbdd.integrate_approx_hint(v2, -16, var, aposteriori=False)
....: _ = dbdd.integrate_approx_hint(v3, 1, var, aposteriori=False)
```

```
integrate approx hint   (conditionning)
      u0 + u4 + u5 + u6 + u8 + ... = -19 + χ(σ²=10.000)
      Worthy hint !   dim=137, δ=1.01322376, β=29.74
integrate approx hint   (conditionning)
     u0 + u5 + u6 + u7 + u8 + ... = -29 + χ(σ²=10.000)
     Worthy hint !   dim=137, δ=1.01329667, β=28.56
integrate approx hint   (conditionning)
     u0 + u4 + u5 + u7 + u12 + ... = -16 + χ(σ²=10.000)
     Worthy hint !   dim=137, δ=1.01337366, β=27.24
integrate approx hint   (conditionning)
     u1 + u2 + u3 + u4 + u5 + ... = 1 + χ(σ²=10.000)
     Worthy hint !   dim=137, δ=1.01340026, β=25.77
```

Here, the cost of the lattice reduction attack has decreased by 3 bikz. While all the hints have been integrated, we finally estimate the security and run the attack again.

```
sage: beta, delta = dbdd.estimate_attack()
....: t = cputime()
....: secret = dbdd.attack()
....: str(cputime(t)) + " CPU Seconds for the attack"
```

```
       Attack Estimation
 dim=137      δ=1.013400      β=25.77

       Running the Attack
Running BKZ-27   Success !

47.93729300000001 CPU Seconds for the attack
```

This time, BKZ stopped at blocksize 27 while the estimation was ≈ 26 bikz.

A.2 Demonstration of the lightweight version

The lightweight implementation is called when the class of the DBDD instance is DBDD_predict. While the heavy basis of the lattice is not stored, only its volume and dimension are stored. Let us create an LWE instance. Before estimating the cost of the lattice reduction attack in bikz, one needs to integrate the q vectors (i.e. drop some LWE samples, see Section 4.5). Then, the security is computed in bikz thanks to the volume and dimension of the lattice.

```
sage: load("../framework/instance_gen.sage")
....: n = 512
....: m = n
....: q = 2 ^ 15
....: D_e = {-2: 0.05, -1: 0.20, 0: 0.5, 1: 0.20, 2: 0.05}
....: D_s = D_e
....: A, b, dbdd = initialize_from_LWE_instance(DBDD_predict, n, q, m, D_e, D_s)
....: _ = dbdd.integrate_q_vectors(q, report_every=20)
....: beta, delta = dbdd.estimate_attack()
```

```
      Build DBDD from LWE
 n=512   m=512   q=32768
       Integrating q-vectors
 [...20]   integrate short vector hint
    Worthy hint !   dim=1024, δ=1.00518248, β=270.72

       Attack Estimation
 dim=1016     δ=1.005183      β=270.69
```

We now create a new LWE instance (necessary because the q vectors should always been included at the end). And, we create 4 vectors and simulate side information. Here we only integrate perfect hints.

```
sage: load("../framework/instance_gen.sage")
....: A, b, dbdd = initialize_from_LWE_instance(DBDD_predict, n, q, m, D_e, D_s)
....: # Simulating hints
....: v0 = vec([1 if i < m / 2 else 0 for i in range(m + n)])
....: v1 = vec([0 if i < m / 2 else 1 for i in range(m + n)])
....: v2 = vec([1 if i < m else 0 for i in range(m + n)])
....: v3 = vec([1 if i < m / 4 else 0 for i in range(m + n)])
....: # Computing l = <vi, s>
....: dbdd.leak(v0), dbdd.leak(v1), dbdd.leak(v2), dbdd.leak(v3)
```

```
    Build DBDD from LWE
 n=512   m=512   q=32768
(33, 9, 56, 12)
```

The hints must be now integrated and we assess the lattice reduction cost. As seen in Remark 30, due to the shape of our vectors **v**, new short vector hints must be integrated.

```
sage: # Integrate hints
....: _ = dbdd.integrate_perfect_hint(v0, 33)
....: _ = dbdd.integrate_perfect_hint(v1, 9)
....: _ = dbdd.integrate_perfect_hint(v2, 56)
....: _ = dbdd.integrate_perfect_hint(v3, 12)
....: M = q * identity_matrix(n + m)
....: V = vec(M[0] - M[1])
....: i = 0
....: while dbdd.integrate_short_vector_hint(V):
....:     i += 1
....:     V = vec(M[i] - M[i + 1])
....: beta, delta = dbdd.estimate_attack()
```

```
 integrate perfect hint   u0 + u1 + u2 + u3 + u4 + ... = 33
       Worthy hint !   dim=1024, δ=1.00519338, β=269.83
 integrate perfect hint   u256 + u257 + u258 + u259 ... = 9
       Worthy hint !   dim=1023, δ=1.00520492, β=268.91
 integrate perfect hint   u0 + u1 + u2 + u3 + u4 + ... = 56
       Worthy hint !   dim=1022, δ=1.00521752, β=268.03
 integrate perfect hint   u0 + u1 + u2 + u3 + u4 + ... = 12
       Worthy hint !   dim=1021, δ=1.00522793, β=267.19
 integrate short vector hint   32768*c0 - 32768*c1 ∈ Λ
    Not sure if in Λ,       Unworthy hint, Rejected.
       Attack Estimation
 dim=1021    δ=1.005228      β=267.19
```

Here, with 4 perfect hints, the cost of the lattice reduction attack has decreased by ≈ 3 bikz. The blocksize may be actually below 267.19 as there may be residual short vector hints.

A.3 Demonstation of the super lightweight version

The super lightweight implementation is called when the class of the instance is DBDD_predict_diag. Here, we assume that the covariance matrix is always diagonal. Let us create an LWE instance. We integrate the q vectors (i.e. drop some LWE samples, see Section 4.5) and compute the security in bikz.

```
sage: load("../framework/instance_gen.sage")
....: n = 512
....: m = n
....: q = 2 ^ 15
....: D_e = {-2: 0.05, -1: 0.20, 0: 0.5, 1: 0.20, 2: 0.05}
....: D_s = D_e
....: A, b, dbdd = initialize_from_LWE_instance(DBDD_predict_diag, n, q, m, D_e, D_s)
```

```
....: _ = dbdd.integrate_q_vectors(q, report_every=20)
....: beta, delta = dbdd.estimate_attack()
```

```
      Build DBDD from LWE
 n=512   m=512   q=32768
       Integrating q-vectors
 [...20]   integrate short vector hint
        Worthy hint !   dim=1024, δ=1.00518248, β=270.72
       Attack Estimation
 dim=1016    δ=1.005183      β=270.69
```

We create 20 canonical (necessary to keep the covariance matrix diagonal) vectors for integrating perfect hints.

```
sage: A, b, dbdd = initialize_from_LWE_instance(DBDD_predict_diag, n, q, m, D_e, D_s)
....: # Simulating hints
....: v = [[] for _ in range(20)]
....: for i in range(20):
....:     v[i] = canonical_vec(m + n, i)
```

```
     Build DBDD from LWE
 n=512   m=512   q=32768
```

The perfect hints are integrated into a new instance and the security is estimated.

```
sage: # Integrate hints
....: for i in range(20):
....:     _ = dbdd.integrate_perfect_hint(v[i], dbdd.leak(v[i]))
....: _ = dbdd.integrate_q_vectors(q, report_every=20)
....: beta, delta = dbdd.estimate_attack()
```

```
 integrate perfect hint   u0 = -2
         Worthy hint !   dim=1024, δ=1.00519245, β=270.02
 integrate perfect hint   u1 = -1
         Worthy hint !   dim=1023, δ=1.00520080, β=269.31
 integrate perfect hint   u2 = 1
         Worthy hint !   dim=1022, δ=1.00520918, β=268.61
 integrate perfect hint   u3 = 0
         Worthy hint !   dim=1021, δ=1.00521757, β=267.91
 integrate perfect hint   u4 = 1
         Worthy hint !   dim=1020, δ=1.00522774, β=267.21
 integrate perfect hint   u5 = 1
         Worthy hint !   dim=1019, δ=1.00523618, β=266.51
 integrate perfect hint   u6 = 0
         Worthy hint !   dim=1018, δ=1.00524465, β=265.81
 integrate perfect hint   u7 = -2
         Worthy hint !   dim=1017, δ=1.00525490, β=265.11
 integrate perfect hint   u8 = 0
         Worthy hint !   dim=1016, δ=1.00526341, β=264.41
```

```
integrate perfect hint   u9 = 0
        Worthy hint !   dim=1015, δ=1.00527195, β=263.71
integrate perfect hint   u10 = 2
        Worthy hint !   dim=1014, δ=1.00528229, β=263.01
integrate perfect hint   u11 = 0
        Worthy hint !   dim=1013, δ=1.00529087, β=262.32
integrate perfect hint   u12 = 0
        Worthy hint !   dim=1012, δ=1.00529947, β=261.62
integrate perfect hint   u13 = 2
        Worthy hint !   dim=1011, δ=1.00530810, β=260.93
integrate perfect hint   u14 = 0
        Worthy hint !   dim=1010, δ=1.00531856, β=260.23
integrate perfect hint   u15 = 2
        Worthy hint !   dim=1009, δ=1.00532723, β=259.54
integrate perfect hint   u16 = 1
        Worthy hint !   dim=1008, δ=1.00533593, β=258.85
integrate perfect hint   u17 = 0
        Worthy hint !   dim=1007, δ=1.00534647, β=258.16
integrate perfect hint   u18 = -1
        Worthy hint !   dim=1006, δ=1.00535522, β=257.46
integrate perfect hint   u19 = 1
        Worthy hint !   dim=1005, δ=1.00536399, β=256.77
      Integrating q-vectors
[...20]   integrate short vector hint   32768*c1023 ∈ Λ
     Worthy hint !   dim=1004, δ=1.00536426, β=256.76
[...20]   integrate short vector hint   32768*c1003 ∈ Λ
     Worthy hint !   dim=984, δ=1.00536755, β=256.54
      Attack Estimation
dim=979     δ=1.005368      β=256.53
```

The integration of 20 perfect hints implies here a loss of ≈ 13 bikz.