A Generic Framework for Side-Channel Attacks against LWE-based Cryptosystems

Julius Hermelink[1], Silvan Streit[2,3], Erik Mårtensson[4,5], and Richard Petri[1]

[1] Max Planck Institute for Security and Privacy, Bochum, Germany, {julius.hermelink,richard.petri}@mpi-sp.org
[2] Fraunhofer AISEC, Garching, Germany, silvan.streit@aisec.fraunhofer.de
[3] Technical University of Munich (TUM), Munich, Germany
[4] Lund University, Lund, Sweden, erik.martensson@eit.lth.se
[5] Advenica AB, Malmö, Sweden

Abstract. Lattice-based cryptography is in the process of being standardized. Several proposals to deal with side-channel information using lattice reduction exist. However, it has been shown that algorithms based on Bayesian updating are often more favorable in practice.
In this work, we define *distribution hints*; a type of hint that allows modelling probabilistic information. These hints generalize most previously defined hints and the information obtained in several attacks.
We define two solvers for our hints; one is based on belief propagation and the other one uses a greedy approach. We prove that the latter is a computationally less expensive approximation of the former and that previous algorithms used for specific attacks may be seen as special cases of our solvers. Thereby, we provide a systematization of previously obtained information and used algorithms in real-world side-channel attacks.
In contrast to lattice-based approaches, our framework is not limited to value leakage. For example, it can deal with noisy Hamming weight leakage or partially incorrect information. Moreover, it improves upon the recovery of the secret key from approximate hints in the form they arise in real-world attacks.
Our framework has several practical applications: We exemplarily show that a recent attack can be improved; we reduce the number of traces and corresponding ciphertexts and increase the noise resistance. Further, we explain how distribution hints could be applied in the context of previous attacks and outline a potential new attack.

Keywords: Lattice-based cryptography · Lattice Reduction · ML-KEM · Belief Propagation · Side-Channel Attacks · Kyber

1 Introduction

The National Institute of Standards and Technology (NIST) post-quantum standardization process is in the fourth round, and the draft standards for four post-quantum schemes that have been published. Two of those schemes base their security on the hardness of the Module Learning with Errors (MLWE)

problem – ML-DSA [19], a signature scheme, and ML-KEM [20], a Key Encapsulation Mechanism (KEM). ML-KEM was known as Kyber throughout the competition [2], and ML-DSA was known as Dilithium [8]. Both ML-DSA and ML-KEM are particularly well-suited for embedded devices, and with the fourth round still in process, ML-KEM is currently the only key exchange scheme that has been selected for standardization. With the standardization of two Learning with Errors (LWE)-based schemes being imminent, understanding their side-channel security is crucial.

A wide variety of attacks on the secret key is already known. Several works have focused on the inverse Number Theoretic Transform (NTT) during the decapsulation [24,11,30,25]. Another line of research have been plaintext-checking, full decryption and decryption failure oracles (see, e.g., [28,23,15,30,26]). In many of these attacks, an algorithm to recover the secret key from the obtained side-channel information is required.

Known classic attacks on lattice-based schemes – unsurprisingly – rely on lattice-reduction. The public key equation allows deriving (computationally hard) shortest vector problems, which give the secret key when solved. These attacks are known as the primal and the dual attack [1]. The framework of [4] explains how the complexity of the lattice resulting from the primal attack can be reduced using side-channel information. In their work, the authors assume that side-channel information is given in terms of several types of hints. These hints can be applied to the Distorted Bounded Distance Decoding Problem (DBDD) posed by the public key equation and reduce the hardness of the subsequently derived unique Shortest Vector Problem (uSVP). Their framework has been extended in [5] to the information arising in [9]. A recent work by May and Nowakowski [18] greatly reduces the complexity of integrating perfect and modular hints on parts of the secret. Their work integrates into the LWE instance instead of working with a derived DBDD problem.

In practice, many side-channel attacks rely on different techniques. Most notably this includes Belief Propagation (BP)[24,22,16,11,15], but different approaches have also been explored [23,27]. In the case of (uncorrelated[6]) decryption failure information, lattice-based approaches have proven to be less efficient than BP (compare [3] with [15] and see [6]). It has also been shown that BP can be combined with lattice reduction in some cases [12]. A recent work published at TCHES 2024 [27] claims that their greedy approach outperforms BP by a factor of two in terms of required information in the context of decryption failure inequalities. These types of algorithms work by iteratively updating guesses or probability distributions, which is why we call them *Bayesian updating-based*.

Current Bayesian updating-based solvers are specific to the targeted information, or even to concrete attacks. While they often perform better in special cases, a systematic and more generally applicable approach complementing lattice-based hints is yet missing. Further, neither the relation between different types of Bayesian-updating-based algorithms nor between those algorithms to lattice-based approaches is yet understood.

[6] These arise in several chosen-ciphertext attacks.

Our contribution. In this work, we define *distribution hints* and state algorithms to solve for the secret key. Our definition of hints entails all but one type of previously defined lattice-based hints in a single generalized form. Moreover, our definition allows modelling the information arising in several attacks in practice that was previously not, or only insufficiently, captured. In particular, uncertainty in side-channel information is intrinsic to our definition.

We propose two algorithms to solve for the secret key from distribution hints: a BP-based algorithm and a greedy approach. We explain the conceptual difference between the greedy algorithm and BP – the greedy solver can be seen as an approximation of the BP. While, in theory, the greedy solver requires more hints for key recovery than BP, it may outperform BP due to numerical problems and gives advantages with regard to practical considerations. This gives a more nuanced perspective on the claim of [27] (see above): Greedy solvers and BP complement each other. Further, we prove that previous algorithms to solve decryption failure inequalities can be seen as special cases of our method.

Compared to lattice-based solvers, our framework allows modelling hints that are incorrect with positive probability. Further, we require fewer approximate hints or may even solve for the secret key for cases in which lattice-based approaches could not achieve a sufficient reduction in computational hardness. For example, we may fully recover an ML-KEM768 secret key from leakage that occurs in practice for which previous work requires running BKZ with $\beta \approx 300$. In addition, our framework covers various new types of leakage, e.g., with non-uniform noise distributions or hints on Hamming weights (HWs).

Our framework has several practical applications: We exemplarily show that it may be used to reduce the required information for key recovery in the attack of [27] by a factor of more than two; this makes the attack practical for noise levels that had previously required greatly increased numbers of traces that quickly became infeasible. Furthermore, we explain how distribution hints could potentially be used to improve upon previous attacks on the inverse NTT [24,11]. We also outline a previously unexplored attack on ML-KEM's decryption routine.

In summary, our work provides a systematic way of dealing with side-channel information, complementing previous work on lattice-based hints. We provide a framework for side-channel information that is far more extensive, flexible, and more generally applicable than previous work.

Conceptual comparison to previous frameworks. As noted in [18], the leaky LWE framework [4] takes a lattice-centric approach to side-channel information by working on a DBDD instance in which the hints are integrated. On the other hand, the framework of [18] takes the LWE-centric route; hints are integrated to the LWE instance. Practical attacks mostly used Bayesian updating-based approaches such as BP or greedy algorithms. While the work of [12] combines lattice reduction with BP, these attacks are fundamentally based on updating probabilities based on information arising from side-channel attacks. In fact, [12] just integrates the BP output into the LWE instance. These algorithms are often superior in the setting of their specific attack. Our framework formalizes and generalizes the approach – complementary to the lattice-based frameworks.

Application to different types of schemes. Our solvers can also be adapted to other classes of schemes. Large parts of the algorithms are not dependent on a LWE or even lattice-based settings. We rely on [12] to integrate BP output into the primal attack lattice derived from the public key equation. Additionally, we show how this can be done for greedy solvers as well. This part is specific to an LWE-based setting, and if it can be adapted to different types of schemes, our method applies as well.

Practical recommendations. Our work together with the work of [18] covers how to efficiently integrate all types of hints defined in [4,5] except short vector hints and modular hints on both $\mathbf{e}$ and $\mathbf{s}$. For error-free modular/perfect hints on $\mathbf{s}$, we recommend the approach of [18]. In the case of approximate hints with uniform coefficients and perfect hints on $\mathbf{x} = (\mathbf{e}, \mathbf{s})$, the framework of [4] has an advantage but requires vast computational resources, while the work of [18] does not yet consider these hints. For inequality hints with coefficients that are uncorrelated to the coefficients of the secret key (arising in [9]), [5] seems to require the lowest number of hints.

For inequality hints with correlated coefficients (arising, e.g., in [23,3,15,6,13]), erroneous perfect and approximate hints, approximate hints with small coefficients or large noise, and whenever leakage on HWs is obtained, our framework should be applied. Furthermore, we provide the first algorithms that may recover the secret key in practice on widely-available hardware for hints on both parts of the secret key. If coefficients are small, we recommend our BP instantiation. In all other cases, the greedy solver should be used.

Open source and optimized implementations. All resources developed for this work are publicly available[7]. This includes a Rust implementation of both a greedy solver and a BP solver, both of which can be compiled to a Python module. Our solvers are optimized for performance and fully multithreaded, and include specialized implementations for types of hints in which computations can be carried out more efficiently. We also provide a simplified Python implementation of the method of [12].

Acknowledgements

We would like to thank the authors of [27], in particular Thales Paiva, for the helpful discussion and the access to their source code. Silvan Streit was supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, in the project "Trusted Electronics Center Bavaria". Erik Mårtensson was funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[7] Available under `https://github.com/juliusjh/distribution_hints` and `https://github.com/juliusjh/distribution_hints_solvers`.

2 Background

After introducing some basic notation, we reiterate the required background on lattice-based hints, i.e., the works of [4,5,18]. We then give a brief overview of BP and its application to side-channel attacks. Finally, we discuss recent solvers for decryption failure inequalities. This includes the belief-propagation-based solver of [12] and the greedy approach of [27].

Notation. Vectors are generally denoted in bold, i.e., as $\mathbf{a}, \mathbf{x}, \ldots$. We will denote a random variable a following a distribution $\mathcal{D}$ as $a \sim \mathcal{D}$. For vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$, we denote the inner product by $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i$. In the following, we assume an MLWE setting in $(\mathbb{F}_q/(f))^k$ with $n = \deg(f)$; that means, for $k = 1$ we have an Ring Learning with Errors (RLWE) setting and for $n = 1$ we have an unstructured LWE setting. Given the secret key $\mathbf{x} = (\mathbf{e}, \mathbf{s}) \in \{-\eta, \ldots, \eta\}^{2kn}$, we use j as index for key coefficients, and i as index for hints. Key guesses will be denoted by $\mathbf{x}'$, while the true key will always be $\mathbf{x}$. We will also implicitly use $\mathbf{x}$ to denote the random variable belonging to the true key, i.e., in the form of $P(\mathbf{x} = \mathbf{x}')$. Let $\mathbf{B}_\eta$ denote the central binomial distribution, that is, a distribution whose output is computed as $\sum_{i=1}^{\eta}(a_i - b_i)$, where a_i and b_i are independently and uniformly randomly sampled from $\{0, 1\}$. Let $\mathcal{N}(\mu, \sigma)$ denote the normal distribution with mean μ and standard deviation σ. A message from variable node i to factor node j at step t of a BP instantiation is denoted by $m_{t,i,j}$ or $m_{i,j}$ if the step is irrelevant. Further, $\mathrm{supp}(\mathcal{D})$ is the support of the distribution, i.e., $\{a \in \mathcal{D} : P(a) \neq 0\}$, and $\mathrm{Dom}(x_j)$ the domain of the secret key coefficients.

2.1 The Learning with Error Problem Family

ML-KEM relies on the MLWE problem, which is a variant of the LWE problem.

Learning with Errors. The LWE problem is defined as finding a coefficient-wise small vector $\mathbf{s} \in \mathbb{F}_q^n$ given samples of the form $\langle \mathbf{a}_i, \mathbf{s} \rangle + e_i = t_i \in \mathbb{F}_q$ for uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}^n$ and $e_i \in \mathbb{F}_q$ sampled from an error distribution χ with small (when interpreted as integers) support. In cryptographic practice, the number of samples is often fixed to the dimension of $\mathbf{s}$ and can thus be written as recovering $\mathbf{s}$ from both $\mathbf{A}$ and $\mathbf{sA} + \mathbf{e} = \mathbf{t}$ where $\mathbf{A} \in \mathbb{F}_q^{n \times n}$, $\mathbf{e}, \mathbf{t} \in \mathbb{F}_q^n$. In practice, both $\mathbf{s}$ and $\mathbf{e}$ are often coefficient-wise sampled from a small, central binomial distribution. Clearly, $\mathbf{s}$, $\mathbf{e}$, and $\mathbf{t}$ can be chosen to be matrices as well – combining several LWE secrets and samples into one matrix.

Ring Learning with Errors. The RLWE problem replaces $\mathbb{Z}_q^n$ by a factor ring of the form $R = \mathbb{F}_q[x]/(f)$ where $f = x^n + 1$ in the case of Newhope, the predecessor of ML-KEM. This means that the adversary is asked to recover $\mathbf{s} \in R$ given both $\mathbf{a}$ and $\mathbf{sa}+\mathbf{e} = \mathbf{t}$ where $\mathbf{a}, \mathbf{e}, \mathbf{t} \in R$. It can be easily seen that this gives an LWE instance: The elements of R can be written as an n-tuple over $\mathbb{F}_q$ with a different multiplication arising from reducing modulo the ideal generated by f.

Now our equation is a polynomial equation over $\mathbb{F}_q$, and writing out the equations coefficient-wise gives exactly n LWE equations. However, these LWE samples are not independently sampled but structured. Whether structured samples are more vulnerable to attacks is a controversially discussed matter, but until now, no attacks have been able to exploit this structure.

The structure also drastically reduces the computational effort of a key exchange relying on the RLWE problem: elements of R can multiplied using a so-called NTT in $O(n \log n)$. The NTT may be seen as a discrete Fast-Fourier Transformation (FFT) and computes the Chinese remainder theorem; pointwise multiplication in the target domain is polynomial multiplication in R.

Module Learning with Errors. The MLWE problem can be seen as a middle ground between LWE and RLWE. Here, $\mathbf{A}$ is an element of $R^{k \times k}$ for some (typically small) integer k, $\mathbf{s}, \mathbf{e} \in R^k$. The adversary is again given $\mathbf{A}$ and $\mathbf{sA} + \mathbf{e} = \mathbf{t} \in R^k$ and asked to find $\mathbf{s}$. For $k = 1$, the MLWE is just an RLWE problem, for setting k to n and n to 1, it is the (unstructured) LWE problem. In ML-KEM, $k \in \{2, 3, 4\}$, $n = 256$, and $q = 3329$. Clearly, an MLWE instance directly gives multiple (structured) LWE samples.

The primal attack. Common approaches to solving (ring/module) LWE include the primal and the dual attack on LWE. Both methods rely on lattice reduction, in most cases using the Blockwise Korkine-Zolotarev (BKZ) algorithm. Current frameworks that recover the secret key from side-channel hints using lattice reduction rely on the primal attack [4,5,12,18].

The primal attack relies on first obtaining a Closest Vector Problem (CVP) from the LWE instance and a Shortest Vector Problem (SVP) from the CVP using Kannan's embedding. Given $\mathbf{A} \in \mathbb{Z}^{n \times m}, \mathbf{t} \in \mathbb{Z}^m$ and unknown $\mathbf{s} \in \mathbb{Z}^n$ and $\mathbf{e} \in \mathbb{Z}^m$ over the integers such that $\mathbf{sA} + \mathbf{e} \equiv \mathbf{t} \mod q$, the matrix

$$\begin{pmatrix} q\mathbf{I}_m & 0 \\ \mathbf{A} & I_n \end{pmatrix} \tag{1}$$

generates a lattice containing $(\mathbf{e}, \mathbf{s})$ which is close to $(\mathbf{t}, \mathbf{0})$. Adding $(\mathbf{t}, \mathbf{0})$ to the lattice (and increasing the dimension by one to avoid solutions of the form $\mathbf{sA} + k\mathbf{e}$ for $k \neq 1$) thus results in a lattice in which $(\mathbf{e}, \mathbf{s}, 0)$ is short.

2.2 ML-KEM

ML-KEM is the only key exchange scheme that has been selected in the third round of the NIST standardization process. It bases its security on the MLWE problem, and its public key equation directly poses an MLWE problem. ML-KEM is constructed by first defining a Public-Key Encryption (PKE) scheme, and then deriving an IND-CCA2-secure KEM using a Fujisaki-Okamoto (FO)-transform. In the following, we give a quick introduction to ML-KEM ignoring the NTT. For the definitions of the compression, decompression and all parameters, we refer to [20]. The PKE and KEM are stated in Figure 1 and Figure 2,

respectively. The parameters are chosen depending on the security level; we have $\eta_1 \in \{2, 3\}$, $\eta_2 = 2$, $n = 256$, and $k \in \{2, 3, 4\}$. For a full description of ML-KEM, see [20].

The PKE. The secret key (of the PKE) is a vector of polynomials $\mathbf{s} \in R^k = (\mathbb{F}_q[x]/(x^n + 1))^k$, with coefficient-wise small values sampled from the central binomial distribution $\mathbf{B}_{\eta_1}$. The public key directly gives an MLWE sample

$$(\mathbf{s}\mathbf{A}^\top + \mathbf{e} = \mathbf{t}, \mathbf{A}) \tag{2}$$

for secret $\mathbf{e} \in R^k$ sampled from $\mathbf{B}_{\eta_1}$, but instead of $\mathbf{A}$, the seed required to sample $\mathbf{A}$ is stored. The key pair of the PKE is generated as described in Algorithm 1.

The PKE receives a parameter r that determines the pseudorandomness and samples $\mathbf{r}_1, \mathbf{r}, \mathbf{e}_2 \in \mathbb{R}^k$ and $e_1 \in R$. The ciphertext components c_1 and c_2 are computed by compressing

$$\mathbf{u} = \mathbf{r}\mathbf{A} + \mathbf{e}_1 \text{ and } v = \langle \mathbf{t}, \mathbf{r} \rangle + e_2 + m_{\text{poly}}, \tag{3}$$

where m_{poly} is the message represented as a polynomial. To obtain m_{poly}, unset bits of the message are mapped to $0 \in \mathbb{F}_q$ and set bits are mapped to $\lceil q/2 \rceil \in \mathbb{F}_q$.

Ignoring errors caused by the compression, the decryption may now retrieve the decrypted message $\mathtt{m}'$ by computing $v' - \langle \mathbf{s}, \mathbf{u}' \rangle$. Denoting the decompressed v and $\mathbf{u}$ as v', $\mathbf{u}'$, respectively, and the differences caused by the compression by $\Delta\mathbf{u}$ and Δv, the decryption arrives at

$$v' - \langle \mathbf{s}, \mathbf{u}' \rangle = m_{\text{poly}} + \langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 \rangle + e_2. \tag{4}$$

Now the coefficients of

$$\langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, (\mathbf{e}_1 + \Delta u) \rangle + \Delta v + e_2 \tag{5}$$

are small. The message m can therefore be recovered with high probability by mapping coefficients to a zero bit if and only if they are closer (when interpreted as symmetrically reduced integers) to 0 than they are to $\lceil q/2 \rceil$. In ML-KEM, the decryption failure probability is smaller than 2^{-139} for all security levels. The term in (4) is called the **noisy message**, and the one in (5) is called the **noise term** or **error term**. If an adversary with access to the ciphertext generation can recover a coefficient of the noise term or the noisy message, they may derive a linear equation on the secret $\mathbf{x} = (\mathbf{e}, \mathbf{s})$ over the integers.

The KEM. The generation of the KEM keypair is described in Algorithm 1; the secret key of the KEM additionally contains the public key, a hash of the public key, and a rejection value z. Apart from that, it is essentially the key generation of the PKE. The encapsulation, see Algorithm 5, samples a random message, derives a shared secret and the coins r, and encrypts the message using the public key and the randomness r to a ciphertext `ct`. The decapsulation

Algorithm 1 PKE.KeyGen

Input: Randomness seeds ρ, σ
Output: Public key $\mathtt{pk}$, secret key $\mathtt{sk}$

1: $\mathbf{A} \in R^{k \times k} \xleftarrow{\$} \mathcal{U}(\rho)$
2: $\mathbf{e}, \mathbf{s} \in R^k \xleftarrow{\$} \mathbf{B}_{\eta_1}(\sigma)$
3: $\mathbf{t} \leftarrow \mathbf{sA}^\top + \mathbf{e}$
4: **return** $\mathtt{pk}_{\text{pke}} = (\mathbf{t}, \rho), \mathtt{sk}_{\text{pke}} = \mathbf{s}$

Algorithm 2 PKE.Dec

Input: $\mathtt{sk} = \mathbf{s}$, $\mathtt{ct} = (c_1, c_2)$
Output: Decrypted message $\mathtt{m}'$

1: $\mathbf{u}', v' \leftarrow \text{Decompress}(\mathtt{ct})$
2: $m_{\text{poly}} \leftarrow v' - \langle \mathbf{s}, \mathbf{u}' \rangle$
3: $\mathtt{m}' \leftarrow \text{Compress}(m_{\text{poly}}, 1)$
4: **return** $\mathtt{m}'$

Algorithm 3 PKE.Enc

Input: $\mathtt{pk} = (\mathbf{t}, \rho)$, $\mathtt{m}$, coins r
Output: Ciphertext $\mathtt{ct} = (c_1, c_2)$

1: $\mathbf{A} \in R^{k \times k} \xleftarrow{\$} \mathcal{U}(\rho)$
2: $\mathbf{r}_1 \in R^k \xleftarrow{\$} \mathbf{B}_{\eta_1}(r)$
3: $\mathbf{r}, \mathbf{e}_1 \in R^k \xleftarrow{\$} \mathbf{B}_{\eta_2}(r)$
4: $e_2 \in R \xleftarrow{\$} \mathbf{B}_{\eta_2}(r)$
5: $m_{\text{poly}} \leftarrow \text{Decompress}(\mathtt{m}, 1)$
6: $\mathbf{u} \leftarrow \mathbf{rA} + \mathbf{e}_1$
7: $v \leftarrow \langle \mathbf{t}, \mathbf{r} \rangle + e_2 + m_{\text{poly}}$
8: $c_1 \leftarrow \text{Compress}(\mathbf{u}, d_u)$
9: $c_2 \leftarrow \text{Compress}(v, d_v)$
10: **return** $\mathtt{ct} = (c_1, c_2)$

Fig. 1: Simplified version of the PKE defined by ML-KEM. Public key equation, ciphertext generation, and noisy message computation are highlighted.

may now arrive at the same shared secret by decryption the PKE ciphertext. However, to ensure IND-CCA2 security, it first reencrypts the message to $\mathtt{ct}'$, and checks whether the $\mathtt{ct}$ was honestly generated by comparing $\mathtt{ct}$ to $\mathtt{ct}'$. In case of a mismatch, the decapsulation is rejected by returning z instead of the shared secret.

Algorithm 4 KEM.KeyGen

Input: Randomness seeds ρ, σ
Output: Public key $\mathtt{pk}$, secret key $\mathtt{sk}$

1: $z \xleftarrow{\$} \mathcal{U}()$
2: $\mathtt{pk}, \mathtt{sk}_{\text{pke}} \leftarrow \text{PKE.KeyGen}()$
3: $h = \text{H}(\mathtt{pk})$
4: $\mathtt{sk} \leftarrow (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$
5: **return** $\mathtt{pk}, \mathtt{sk}$

Algorithm 5 KEM.Encaps

Input: $\mathtt{pk}$
Output: Ciphertext $\mathtt{ct}$, shared secret $\mathtt{K}$

1: $\mathtt{m} \xleftarrow{\$} \mathcal{U}()$
2: $\bar{\mathtt{K}}, \mathtt{r} \leftarrow \text{G}((\mathtt{m}, \text{H}(\mathtt{pk}))$
3: $\mathtt{ct} \leftarrow \text{PKE.Enc}(\mathtt{pk}, \mathtt{m}, \mathtt{r})$
4: $\mathtt{K} \leftarrow \text{KDF}((\bar{\mathtt{K}}, \text{H}(\mathtt{ct})))$
5: **return** $\mathtt{ct}, \mathtt{K}$

Algorithm 6 KEM.Decaps

Input: $\mathtt{sk} = (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$, $\mathtt{ct}$
Output: Shared secret $\mathtt{K}$

1: $\mathtt{m}' \leftarrow \text{PKE.Dec}(\mathtt{sk}_{\text{pke}}, \mathtt{ct}_{\text{pke}})$
2: $\bar{\mathtt{K}}', \mathtt{r}' \leftarrow \text{G}((\mathtt{m}', h))$
3: $\mathtt{ct}' \leftarrow \text{PKE.Enc}(\mathtt{pk}, \mathtt{m}', \mathtt{r}')$
4: $b \leftarrow \text{Compare}(\mathtt{ct}, \mathtt{ct}')$
5: **if** b **then**
6: **return** $\mathtt{K} = \text{KDF}(\mathtt{K}', \text{H}(\mathtt{ct}))$
7: **else**
8: **return** $\mathtt{K} = \text{KDF}(z, \text{H}(\mathtt{ct}))$
9: **end if**

Fig. 2: Simplified version of ML-KEM. G and H are hash functions.

2.3 Lattice Reduction-Based Hints

The most promising algorithms to solve LWE instances rely on lattice reduction techniques. To recover the secret key, several methods that also rely on lattice reduction have been suggested. Side-channel information is assumed to be available in the form of "hints" that can be used to reduce the hardness of the resulting SVP instance in the primal attack. If the hardness of the SVP has been sufficiently reduced, the secret key can be obtained using lattice reduction.

The framework of [4]. The framework of [4] defines four types of hints that allow reducing the hardness of an LWE instance. To make use of these hints, the authors suggest to first derive a DBDD instance from the LWE instance. The DBDD instance can in turn be converted to an SVP instance similar to the one derived in the primal attack. Hints are "integrated" into the DBDD instance, reducing the computational complexity of the subsequently obtained SVP.

Let $\mathbf{x} = (\mathbf{e}, \mathbf{s})$ denote the LWE secrets, $\mathbf{v}$ denote a known vector and l denote a scalar value. Furthermore, the parameters k, σ and the lattice Λ are assumed to be known to the attacker. The following hints are defined in [4]:

- **Perfect hint**: $\langle \mathbf{v}, \mathbf{x} \rangle = l$
- **Modular hint**: $\langle \mathbf{v}, \mathbf{x} \rangle = l \mod k$
- **Approximate hint**: $\langle \mathbf{v}, \mathbf{x} \rangle = l + \mathcal{N}(0, \sigma)$
- **Short vector hint**: $\mathbf{v} \in \Lambda$

While powerful for estimations and theoretical considerations, this method of dealing with side-channel information has proven to be difficult to apply in practice due to its long runtime in high dimensions. Further, it has been shown that using BP may outperform the lattice-based approach in terms of required information for full key recovery [6]. This is at least the case for the type of decryption failure inequalities arising in attacks such as, e.g., [23,3,15] (c.f., Section 2.5). In these attacks, the adversary obtains an inequality over the noise term (5) and thereby over the secret key. In contrast to the attacks such as, e.g., [9], the inequality coefficients are correlated with the secret key. (Also in the case of uncorrelated decryption failure inequalities, the framework has proved to be impractical [9].)

The work of [5] extends the framework of [4] by adding hints that aim at the uncorrelated decryption failure inequalities of [9]. While drastically improving upon the practicality for uncorrelated inequalities, to the best of our knowledge, the method is not well suited for correlated inequalities.

The improvements of [18]. In [18] methods for efficiently integrating perfect hints and modular hints are introduced. For modular hints, their method simply corresponds to reducing the dimension by the number of hints given. However, for perfect hints, their approach decreases the difficulty of the underlying LWE problem by more than one dimension per hint. They build up a so-called hint matrix, where each column corresponds to the parameters $[\mathbf{v}, l]^T$ of a perfect hint.

The larger the (absolute value of the) determinant of the lattice spanned by these columns, the easier the transformed lattice problem becomes. If the columns $[\mathbf{v}, l]^T$ consist of uniformly random vectors from $\mathbb{F}_q^{n+1}$, then the determinant becomes very large and their approach performs surprisingly well. The approach of [18] does not cover any of the other hints introduced by [4,5].

2.4 Belief Propagation

BP is a message passing algorithm that aims at computing marginal distributions. It was first described in [10] to decode Low-Density Parity Check (LDPC) codes, and has been proposed for recovering the secret key in side-channel attacks in the seminal work of [29]. In [15], it was shown that BP can be used to solve for the secret key from decryption failure information. Their instantiation makes use of a fully connected graph which is very unusual; however, due to the structure of the processed information, computational improvements first presented in [23] allow for a somewhat efficient computation. The work of [12] explains how BP may be combined with lattice reduction.

Let $(X_0, \ldots, X_{n-1}) = \mathbf{X}$ be a vector of random variables with joint mass function $p(\mathbf{X})$. Let $f_i : I_i \to [0, 1] \subseteq \mathbb{R}$ such that $p(\mathbf{X}) = \prod_{i=0}^{n'} f_i(I_i)$ for $n' < n$ and $I_i \subseteq \{X_0, \ldots, X_{n-1}\}$. Then, BP aims at computing the marginals for $p(\mathbf{X})$, i.e., $p(X_j)$ for all j, using a factor graph consisting of variable nodes for the X_j and factor nodes that represent the relationship between variable nodes, i.e., the f_i. An edge between a variable node with index j and a factor node with index i represents that f_i has X_j as input.

BP passes messages from variable nodes to factor nodes and vice versa. Messages represent probability distributions for a variable node. Variable nodes have an associated prior; in the first step, the variable nodes send the priors to the factor nodes. Given message $m_{t,i,j}$, i.e., messages from variable node i to factor node j at step t, the factor nodes compute the update for the i'-th variable node as

$$m_{t+1,j,i}(x) = \sum_{\mathbf{x}, x_i = x} f_j(\mathbf{x}) \prod_{i' \neq i} m_{t,i',j}(x).$$

In turn, the variable nodes compute the update for the j-th factor node as $m_{t+1,j',i} = \prod_{j=1}^{j'-1} m_{t,i,j} \cdot \prod_{j=j'+1}^{k} m_{t,i,j}$. After every step, i.e., after passing from variable nodes to factor nodes and vice versa, the marginals can be computed by normalizing $b_i(x) = \prod_j \mu_{j,i}(x)$.

2.5 Noise Term Leakage

Recall from (2.2) that the noise term (5), which occurs during the decryption, contains information about the secret key. In fact, as vector, a coefficient of the noise term can be written as

$$\langle \mathbf{v}_i, \mathbf{x} \rangle + c_i \tag{6}$$

where $\mathbf{x} = (\mathbf{e}, \mathbf{s})$ interpreted as integer vectors, $\mathbf{v}_i \in \mathbb{Z}^{2kn}$, and $c \in \mathbb{Z}$. The terms $\mathbf{v}_i, c_i$ are known to an adversary that has access to the ciphertext generation.

Several recent attacks, e.g., [23,3,15,6,14], exploit information contained in the noise term by causing and observing decryption failures. A recent attack also suggested targeting the noise term directly by performing a side-channel analysis on the computation of the noisy message (4) [27].

Decryption failure attacks. Decryption failure attacks introduce an error in the noisy message using either a chosen ciphertext [3,6,14], a fault [23], or a combination of both [15]. This error causes the message recovery (see Section 2.2) to fail if the noise term is positive. If a chosen ciphertext is used, the adversary requires a side-channel to observe whether a failure is caused or the *decryption* succeeds; the *decapsulation* always fails. If a fault or a combination of a chosen ciphertext and a correcting fault is used, it suffices to observe the decapsulation outcome – the fault circumvents the FO-transform. From each observed decryption failure, the adversary obtains an inequality over the noise term and may derive an integer inequality over the secret key of the form (6).

The attack of [27]. The work of [27] suggests targeting the noise term directly during the computation of the subtraction in $\mathbf{v} - \langle \mathbf{u}, \mathbf{s} \rangle$. Thereby, the adversary obtains a probability distribution on the HW of the noisy message. The authors then explain how to derive inequalities in the form of (6) from these distributions. These can be solved using the algorithms presented in [15,7,12]. Additionally, the authors present a new solver.

Solving inequalities using BP. Pessl and Prokop [23] first presented a method based on Bayesian updating to solve for decryption failures. Subsequently, Hermelink, Pessl, and Pöeppelmann [15] used BP to improve upon the number of required inequalities to recover the secret key. Both methods fail if insufficiently many inequalities are available, and no attempt to reduce the computational hardness of the underlying lattice problem is made. Therefore, in contrast to the framework of [4], these solvers do not give any estimates on the remaining hardness to solve for the secret key with a certain number of inequalities. In addition, the information available to the adversary through the public key equation is not considered at all, and no incorrect inequalities may be present in the data set. To circumvent these limitations, the work of [12] explains how to deal with incorrect inequalities. It shows how to use the BP's output to derive an SVP instance from the public key equation (2) that is computationally easier than in the primal attack.

The BP graph used in the method of [15,12] is fully connected, i.e., each variable node is connected to every factor node. Variable nodes represent unknown key coefficients of $\mathbf{x} = (\mathbf{e}, \mathbf{s})$, and factor nodes represent inequalities. The variable nodes are initialized to $\mathbf{B}_{\eta_1}$, the distribution the key coefficients are sampled from, which represents the initial belief into the key. In the first step, the variable nodes send these prior distributions to the factor nodes. A factor node representing an inequality $\sum_i a_i x_i \leq b$ performs the following computation

to update. First, for each coefficient with index i, it computes the distribution of the sum $s_{i'} = \sum_{i \neq i'} a_i x_i$ based on the received beliefs. Then, for each possible value $c \in \{-\eta, \ldots, \eta\}$, the factor nodes compute the updated beliefs as

$$\mathrm{P}(x_i = c) = p\,\mathrm{P}(x_i = c \mid s_i \leq b - c) + (1 - p)\,\mathrm{P}(x_i = c \mid s_i > b - c). \quad (7)$$

where p is the probability of the inequality being correct. These beliefs are sent back to the variable nodes.

Even if the BP cannot fully recover the secret key, some key coefficients can often be assumed to be fully recovered. The authors of [12] suggest sorting coefficients by entropy of the corresponding variable node and argue that all coefficients up to the first incorrect should be assumed to be known to the adversary (c.f., with Theorem 4). Alternatively, an adversary could only assume coefficient with zero entropy to be fully recovered. Every fully recovered coefficient of $\mathbf{s}$ allows reducing the dimension of the LWE instance by one. Every fully recovered coefficient of $\mathbf{e}$ allows deriving a linear equation on one coefficient of $\mathbf{s}$ that can in turn be used to reduce the dimension of the LWE problem. The information on the remaining coefficients comes in form of probability distributions. The most likely key according to these distributions serves as a new target vector for the CVP, further reducing the computational hardness.

The greedy approach of [27]. The attack of [27] requires solving for the secret key from decryption failure inequalities as well. The authors avoid the computationally more expensive BP and present their own greedy approach. They claim that their solver also requires fewer inequalities than BP based methods. This claim stems from the comparison to [7]. However, [7] approximates several computations to reduce the runtime, which also leads to an increased number of inequalities that are required to recover the secret key.

Given a number of linear inequalities $\langle \mathbf{v}_i, \mathbf{x} \rangle \leq b_i, i \in \{1, \ldots, r\}$. Also assuming that we have knowledge about the distribution of the values in $\mathbf{x}$. Let $\mathbf{v}_i = (v_{i,j})_{j \in \{1,\ldots,2kn\}}$, let $\mathbf{x}'$ denote the current guess for $\mathbf{x}$, and define an action (j, c) to be a tuple $\{1, \ldots, 2kn\} \times \mathbb{Z}$ such that $x'_j + c \in \mathrm{Dom}(x_j)$. The greedy solver from [27] then works as follows:

1. Initialize the current solution as the most likely guess $\mathbf{x}'$ and choose an initial value for κ, which is the number of indices to be updated.
2. For hint i, coefficient j, and changes c, compute the score $s_{i,j}(c)$ for the action (j, c) (which modifies $\mathbf{x}'[j]$ to $\mathbf{x}'[j] + c$) as $\max(\langle \mathbf{v}_i, \mathbf{x}' \rangle + v_{i,j}c - b_i, 0)$.
3. Compute the overall score $s_j(c)$ for the action (j, c) as the sum over the scores $s_{i,j}(c)$ from all hints.
4. Perform the best κ actions (j, c) to $\mathbf{x}'$ and adjust κ for the next step.
5. Repeat from Step 2 until a correct solution is found.

The key point here is that in Step 2 for a given action (j, c) and a hint i, the action is scored as 0 if the inequality is fulfilled, and as the distance from being fulfilled otherwise. Also notice that the relative score of an unfulfilled inequality can be seen as $\langle \mathbf{v}_i, \mathbf{x}' \rangle + v_{i,j}c$, as b_i is independent of j and c.

3 Distribution Hints

The framework of [4,5] and the work of [18] already explain how to deal with side-channel information by integrating it into the underlying lattice problem. However, several works, e.g., [24,22,11,23,15,12,27], rely on BP, Bayesian updating, or a greedy approach instead. In this section, we first discuss the limitations of lattice-based hints, and then explain how to generalize hints by *distribution hints*.

3.1 Limitations of Lattice-Based Hints

In previous side-channel attacks, the main limitations of lattice-based hints have been *information loss* and *uncertainty* when modeling obtained information as hints. Information loss occurs if the data that needs to be expressed as hints cannot be transformed without reducing its information content. Uncertainty in obtained data is natural when considering side-channel information – information usually comes in form of a probability distribution on some value or requires a correctness probability to be considered. The former can be modeled using approximate hints; however, approximate hints work with value distributions while information often comes as a distribution on Hamming weights. The transformation from Hamming weight distributions to value distribution hints may in turn cause information loss.

Information loss from conversion. The prime example for information loss is when having to model Gaussian distributions on Hamming weight leakage as Gaussian distributions on values. Side-channel attacks commonly obtain information on Hamming weights that can often be modeled as a Gaussian distribution. Clearly, this gives a probability distribution on the corresponding values, in some cases a prior has to be considered. However, modeling these values as *Gaussian*'s on values, cannot be done without losing most information.

Another example is the case of decryption failure information. These may be modeled as approximate hints, as shown in [4] and [3], but it can be shown (e.g., mentioned in [6]) that the BP of [15,12] requires far fewer inequalities.

Uncertainty in information. In some cases, side-channel information cannot be modeled as a hint at all because the uncertainty cannot be expressed in lattice-based hints. For example, an adversary may obtain a perfect hint, but this hint is only correct with a certain probability $p < 1$ (see Section 5.1). The same problem occurs in the case of decryption failure attacks – a fault or side-channel may often only cause or detect decryption failures with a certain probability p. Then, in addition to potential information loss, the uncertainty cannot be modeled using lattice-based hints.

3.2 Distribution Hints

Given the limitations of lattice-based hints outlined in the previous section, we now define a generalization that avoids these problems. Distribution hints allow

modelling all but one of the kinds of hints named above, formalize the approaches taken in several previous works, and are compatible with a BP-based approach. The public key equation (2) can be made use of by integrating the BP output into the lattice instance following [12]. Let in the following $\mathbf{x} = (\mathbf{e}, \mathbf{s}) \in \mathbb{Z}^{2n}$ be the secret of the LWE instance.

Definition 1. *Let $\mathbf{v} \in \mathbb{R}^{2kn}$, $\mathcal{D}$ be a (discrete) probability distribution on $\mathbb{R}$, such that*

$$\langle \mathbf{v}, \mathbf{x} \rangle \sim \mathcal{D}. \tag{8}$$

We then call the tuple $(\mathbf{v}, \mathcal{D})$ a distribution hint*.*

It should be noted that the definition does not restrict the probability distribution $\mathcal{D}$ or $\mathbf{v}$ in any form. However, in practice, solving distribution hints using BP is limited to cases where $\mathbf{v}$ is relatively small in the 1-norm. We discuss a greedy solver without these limitations in Section 4.2.

3.3 Expressing Lattice-Based as Distribution Hints

Distribution hints may not only model previously defined lattice-based hints except for short vector hints, but also model side-channel information occurring in practice in previous attacks. In the following, we denote the resulting distribution hint as $(\mathbf{v}, \mathcal{D})$.

Proposition 1. *Distribution hints allow expressing perfect, approximate, and modular hints.*

Proof. A **perfect hint** of the form $\langle \mathbf{v}_p, \mathbf{x} \rangle = l$ can simply be expressed with $\mathbf{v} = \mathbf{v}_p$ and $\mathcal{D} = \{l : 1\}$. An **approximate hint** $\langle \mathbf{v}_a, \mathbf{x} \rangle = l + \mathcal{N}(0, \sigma)$ for some standard deviation σ can naturally be written using $\mathbf{v} = \mathbf{v}_a$ and $\mathcal{D} = \mathcal{N}(l, \sigma)$, and it can be seen clearly, that our definition is a generalization. A **modular hint** $\langle \mathbf{v}_m, \mathbf{x} \rangle \equiv l \mod m$ can be expressed as $\mathbf{v} = \mathbf{v}_m$, with $\mathcal{D}$ being the distribution that is uniform on all values that are l modulo m and 0 everywhere else.

Proposition 2. *Distribution hints allow expressing decryption failure inequalities as well as noise term leakage.*

Proof. **Decryption failure inequalities** $\langle \mathbf{v}_{\leq}, \mathbf{x} \rangle \leq b$ can be modeled with the inequalities coefficients being $\mathbf{v} = \mathbf{v}_{\leq}$ and $\mathcal{D}$ the distribution that is uniform on all values smaller or equal than b and 0 everywhere else. **Noise term leakage**, can be expressed in a similar fashion, just that the distribution $\mathcal{D}$ directly expresses the obtained leakage on the noise term.

For example, given a measurement of $\langle \mathbf{v}, \mathbf{x} \rangle$ following a HW distribution $\mathcal{D}_{\mathrm{HW}}$, we set $\mathcal{D}$ to be the corresponding value distribution weighted with the appropriate noise term prior (i.e., considering that the noise term is non-uniform).

4 Solving Distribution Hints

The use of a generalized definition is limited unless these hints can also be solved. In the following, we discuss two different (but related) approaches to solving distribution hints, i.e., obtaining the secret key $\mathbf{x}$ from several hints.

4.1 Solving Distribution Hints using BP

We now give a BP instantiation for solving for the secret key. While our algorithm in theory works for arbitrary distribution hints, it is limited to hints with small $\mathbf{v}$ in practice due to increasing computational complexity. In addition, depending on the types of hints, it may run into numerical problems for instances with a large number of hints. Our algorithm itself is a generalization of the BP used to solve decryption failures in [15,12]. Instead of factor nodes representing inequalities, our factor nodes represent any distribution hint. Thus, the main difference to [15,12] and the dependence of the algorithm on distribution hints, lies in the computations in the factor nodes.

Graph, variable nodes, priors, and messages. Similar to [15,12], our graph consists of a variable node for every unknown key coefficient and a factor node for every hint. A message is a belief (can be thought of as a probability distribution) about the value of a single coefficient of the secret $\mathbf{x}$. Initially, all messages at variable nodes are set to the binomial distribution $\mathbf{B}_\eta$ that the secret key coefficients are sampled from in ML-KEM.

A hint is represented by a single factor node that takes care of updating the belief into all coefficients according to this hint based on the current belief, i.e., based on the messages arriving at the factor node. The variable nodes in turn combine the beliefs coming from the factor nodes. This means that a variable node i receives beliefs from all factor nodes concerning the ith coefficient of $\mathbf{x}$. The message from a variable node with index i to a factor node with index j is the product over all beliefs on x_i leaving out the jth message. Thus, in the next step, the factor node updates the probabilities for all coefficients of $\mathbf{x}$ based on the combined beliefs from all other factor nodes.

Factor nodes Fix a hint $(\mathbf{v}, \mathcal{D})$, i.e., we have $\langle \mathbf{v}, \mathbf{x} \rangle \sim \mathcal{D}$, where $\mathbf{x} = (\mathbf{e}, \mathbf{s})$ is the secret, and a step $t+1$. For each coefficient j, there is precisely one message coming from the j'-th variable node, $m_{t,j',i}$ representing the current belief in the j-th coefficients of $\mathbf{x}$. To compute the outgoing messages $m_{t+1,k,i}$ directed at variable node k, we fix one coefficient and assume the others to follow the distribution given by the respective incoming message $m_{t,j,i}$.

Theorem 1. *Given the hint $(\mathbf{v}, \mathcal{D})$, i.e., $\langle \mathbf{v}, \mathbf{x} \rangle \sim \mathcal{D}$, and messages $m_{t,j,i}$ for some step t, coefficient indices j, and factor node index i, the probability of x_k*

being x'_k, $k \in \{1, \ldots, 2kn\}$, *is proportional to*

$$P(x_k = x'_k \mid x_j \sim m_{t,j,i} \forall j \neq k) \tag{9}$$

$$\propto \sum_{a \in \mathrm{supp}(\mathcal{D})}^{\eta} P_{\mathcal{D}}(a) P(\sum_{j \neq k} v_j x_j = a - v_j x'_j \mid x_j \sim m_{t,j,i} \forall j \neq k). \tag{10}$$

Proof. We have that

$$P(x_k = x'_k) = \sum_{a \in \mathrm{supp}(\mathcal{D})} P(x_k = x'_k | \sum_j v_j x_j = a) P(\sum_j v_j x_j = a). \tag{11}$$

As clearly

$$P(\sum_j v_j x_j = a | x_k = x'_k) = P(v_k x'_k + \sum_{j \neq k} v_j x_j = a) \tag{12}$$

and $P(\sum_j v_j x_j = a) = P_{\mathcal{D}}(a)$, we get

$$P(x_k = x'_k) = \sum_{a \in \mathrm{supp}(\mathcal{D})} P(\sum_{j \neq k} v_j x_j = a - v_k x'_k) P_{\mathcal{D}}(a), \tag{13}$$

from which the statement follows as the prior, i.e., $x_j \sim m_{t,j,i} \forall j \neq k$ only occurs in the first factor of (11) .

Accordingly, for each coefficient with index k of $\mathbf{x}$ the corresponding factor node first computes the probability distribution of

$$s_k = \sum_{j \neq k} v_j x_j \tag{14}$$

from the current belief, i.e., from the incoming messages. This can be done efficiently in the Fourier domain using techniques similar to [23,15]. The factor node then computes the updated belief for the jth coefficient as

$$m_{t+1,j,i}(x') = \sum_{a \in \mathrm{supp}(\mathbf{B}_\eta)} P_{\mathcal{D}}(a) P(s_j = a - v_j x') \ \forall x' \in \mathrm{Dom}(()x_j) \tag{15}$$

where $m_{t+1,j,i}$ is the computed updated message and $\mathrm{Dom}(x_k)$ is the domain for secret key coefficients.

Using the public key equation. The work of [12] explains how to integrate the BP output in the underlying instance. Their work is directly applicable to our approach, and we employ their techniques.

Limitations. Unfortunately, our BP instantiation cannot solve for the secret key from arbitrary distribution hints, but is limited to cases where $\mathbf{v}$ is rather small and not too many hints in total are present. We note that the first condition is often not a restriction, as coefficients in side-channel attacks on lattice-based KEM are often small; this is because otherwise reductions occur that greatly decrease the information content and thus large equations are often not of use in the first place. Our greedy solver avoids some of these limitations.

Large $\mathbf{v}$. The computation in (14) can be carried out efficiently if $\mathbf{v}$ has reasonably small coefficients as shown in [15]. However, their computation using the two-tree approach from [23] has some disadvantages, and we replace it by a simple two-pass algorithm. Nevertheless, if $\mathbf{v}$ has large coefficients, the s_j can take on very large coefficients and the computation of the probability distribution of s_j in (14), becomes inefficient and the memory consumption increases. Therefore, our method is currently limited to cases where $\mathbf{v}$ is reasonably small. In our experiments, the sum of s_j was usually limited to value ranges of length of up to 2^{14} coefficients, each represented by a 64 bit float.

An approach for larger $\mathbf{v}$ could be to first only determine the range of coefficients, i.e., compute the probabilities for intervals of coefficients. As soon as intervals can be ruled out as their probability is close to zero, the intervals for certain coefficients can be refined. This essentially corresponds to iteratively computing upper and lower bounds until the probabilities for actual values can be computed. Thereby, the computed distributions for s_j stay small at all times, as only the probabilities for a certain range have to be computed. We did not implement such an approach, as we are not aware of any attack that requires large $\mathbf{v}$ in this context, and because our greedy solver presented in the next section does not share this limitation with the BP.

Number of hints. The number of messages arriving at a variable node is precisely the number of factor nodes, i.e., the number of hints. To compute the message for factor node i, these messages are (pointwise) multiplied with each other, omitting the incoming message i. Thus, for m hints, the variable nodes compute the product of $m-1$ floats for each potential value of each coefficient. If m is too large and many messages are close to uniform, numerical errors prevent correct updating. This limitation can be circumvented by a determined attacker: using larger floats (we use 64-bit) decreases performance, but this may in turn be circumvented by more cores (scales up to m threads). Moreover, ignoring messages that hold very little information could improve upon this problem by reducing the number of beliefs that are being multiplied. However, in the cases with a large number of hints or large $\mathbf{v}$, it is most likely beneficial to employ the greedy solver introduced in the next section.

Outliers with respect to $\mathcal{D}$. In some cases, an attack will produce hints where the actual value of $\langle \mathbf{v}, \mathbf{x} \rangle$ is rather unlikely and other values are far more probable. In these cases, our solver usually fails. We suspect that in the factor node, the probability for the true value is **numerically** zero.

Filtering out these values leads to convergence, but in most attack scenarios, this will require knowledge of the secret key. A practical way of dealing with this phenomenon is to artificially increase the probability of very unlikely values with non-zero probability in $\mathcal{D}$. This can be done by convoluting the prior with the uniform distribution on an interval. While this theoretically decreases the information content, in a real-world setting, it can prevent the BP from failing. For the greedy solver presented in the next section, this improves upon convergence as well.

4.2 A Greedy Approach to Solving Hints

The work of [27] proposes to use a greedy approach instead of BP in the context of solving decryption failure inequalities. We explain how a greedy approach can be used to solve distribution hints as well. When applying the solver to decryption failure inequalities, the solver of [27] is (almost) a special case of our solver. Finally, we show that both greedy solvers can be considered an approximated version of the BP-based approach.

The greedy solver. Let $(\mathbf{v}_i, \mathcal{D}_i)$ be distribution hints, $\mathbf{v}_i = (v_{i,j})_{j\in\{1,\dots,2kn\}}$, denote the current guess for $\mathbf{x}$ as $\mathbf{x}'$, and define an action (j, c) to be a tuple $\{1, \dots, 2kn\} \times \mathbb{Z}$ such that $x'_j + c \in \mathrm{Dom}(x_j)$. Our greedy solver for distribution hints works as follows:

1. Initialize the current guess with the most likely key $\mathbf{x}'$ and choose an initial value for κ.
2. For hint i, coefficient j, and all potential changes c compute the score $s_{i,j}(c)$ for the action (j, c) as
$$s_{i,j}(c) = \sum_{a\in\mathrm{supp}\,\mathcal{D}_i} P_{\mathcal{D}_i}(a)|\mathbf{v}_i^\top \mathbf{x}' + v_{i,j}c - a|. \tag{16}$$
3. Compute the overall score $s_j(c)$ for the action (j, c) as sum over the scores $s_{i,j}(c)$ from all hints.
4. Perform the best κ actions (j, c) (at most one per coefficient) to $\mathbf{x}'$ and adjust k for the next step.
5. Repeat from Step 2 until the correct solution is found.

In step 4, we select the best action for each coefficient and then only perform one action per coefficient. Clearly, performing several actions for the same coefficient would have already been expressed (and been assigned a score) in another single action. We empirically chose to adjust k by starting with $k = 2n$ and halving it in every step, i.e., $k = \left\lceil \frac{2n}{2^{i \bmod \log_2(2n)}} \right\rceil$. Note that fixed values for k lead to less favorable convergence properties.

Relation to [27]. The solver of [27] (see Section 2.5) is only defined for decryption failure inequalities. These, however, can be modeled as distribution hints, see Lemma 2. Applying our greedy solver to these inequalities yields an algorithm that is very close to the algorithm of [27]. To see that this is close to a generalization of [27] for inequalities $\mathbf{v}_i^\top \mathbf{x} \le b_i$: set $\mathcal{D}_i$ to be the distribution that is 0 for all values smaller than b_j and uniform otherwise:

Theorem 2. *For an inequality $\langle \mathbf{v}_i, \mathbf{x}\rangle \le b_i$, a coefficient j, current guess $\mathbf{x}'$, the changes with best score are the same for both algorithms if no change with score 0 in the algorithm [27] exists.*

Proof. Let $s = \mathbf{v}_i^\top \mathbf{x}' + v_{i,j}c$. Then $s > b_i$ as no change has score 0. Then, because $s > a$ for all $a \in \operatorname{supp} \mathcal{D}_i$, our algorithm computes

$$s_{i,j}(c) = \sum_{a \in \operatorname{supp} \mathcal{D}_i} P_{\mathcal{D}_i}(a)|s - a| = \frac{1}{|\operatorname{supp} \mathcal{D}_i|} \sum_{a \in \operatorname{supp} \mathcal{D}_i} (s - a) \tag{17}$$

$$= s - \frac{1}{|\operatorname{supp} \mathcal{D}_i|} \sum_{a \in \operatorname{supp} \mathcal{D}_i} a \tag{18}$$

where the sum is independent of j and c. Therefore, the relative score is simply s as in the algorithm of [27] for all coefficients, where all scores are non-zero.

For changes which do fulfill the inequality, our algorithm ranks those further away from the bound higher, while [27] does not differentiate between those. Changes very close to the boundary might actually not fulfill the inequality because we only used an approximation for the distribution of the partial sum (14).

Relation to belief propagation. Both BP and the greedy algorithms compute how changes affect the likeliness of a solution with respect to the hints. This asks for whether we can conceptually understand where the differences lie. In fact, the BP and the greedy solver are closely related: by using an approximation, the greedy algorithm avoids costly computations of probability distributions:

Theorem 3. *Our greedy solver is the algorithm that results from replacing the sum distribution by the distance to the most likely value (of the sum) and collapsing the probability distributions to the most likely value after each step and updating only a limited number of variables.*

Proof. For a value $x'_j + c$, a factor node updates the belief by computing the distribution of $v_j c + v_j x'_j + \sum_{j' \neq j} v_{j'} x_{j'}$ by computing

$$\sum_b P_{\mathcal{D}}(v_j c + v_j x'_j + b) P_{\text{sum}}(b) = \sum_a P_{\mathcal{D}}(a) P_{\text{sum}}(a - v_j c + v_j x'_j) \tag{19}$$

for given probability distributions on all $j' \neq j$ where P_{sum} denotes the measure belonging to the distribution of $\sum_{j' \neq j} v_{j'} x_{j'}$. The greedy solver simply approximates $\sum_{j' \neq j} v_{j'} x_{j'}$ by its most likely value, i.e., $\sum_{j' \neq j} v_{j'} x'_{j'}$ for the current best guess $\mathbf{x}'$ and computes $\sum_a P_{\mathcal{D}_i}(a)|\mathbf{v}_i^\top \mathbf{x}' + v_{i,j}c - a|$. In addition, the "variable nodes" do not keep track of scores, but instead collapse to the most likely value, i.e., choose the values with the lowest scores to update the current guess.

While this approximation greatly increases the performance in terms of computational effort, it also decreases the performance in terms of required number of hints. Note that the first two limitations discussed in Section 4.1 do not apply for the greedy solver. Thus, the greedy solver is perfectly suited for situations where $\mathbf{v}$ is large, many hints are present, or a very fast (and easy to implement) solver is required. In this sense, BP and greedy solver complement each other; both are in turn complementary to lattice-based solvers.

Using the public key equation. The work of [12] shows how to use the BP output to decrease the computational hardness of the lattice problem that can be derived from the public key equation (2) in the primal attack. We show that the outputs of our greedy algorithm can be used similarly: The probabilities computed during the BP corresponds to action scores in the greedy algorithm. In fact, the action scores are approximately proportional to the beliefs, as shown in Theorem 3. Thus, the score for no change, i.e., the score $s_{j,0}$, is an (inverse) approximation of the likeliness of the guess for x_j being correct. Using this metric, we may therefore define *correct* and *recovered* similarly to the BP setting:

Definition 2. *Given a key $\mathbf{x} = (x_1, \ldots, x_n)$, current guess $\mathbf{x}'$, and scores $s_{j,0}$ for $j \in \{1, \ldots, n\}$, we call the jth coefficient correct if $x_j = x'_j$. Let w.l.o.g. $s_{j,0} \leq s_{j+1,0}$ for all j. We call a coefficient with index j recovered if $x_{j'} = x'_{j'}$ for all $j' \leq j$.*

We may assume that the adversary knows the number of recovered coefficients using an argument first stated informally in [12] for BP-based solvers. We formalize these arguments and state them in the context of our greedy solver:

Theorem 4. *Let $\mathcal{A}$ be an algorithm that recovers the secret key and checks for its correctness from the guess and the recovered coefficients given by the greedy solver, and let $2kn$ be the dimension of the secret. Given the scores for zero actions for every coefficient, we may then recover the secret without knowledge of the recovered coefficient with at most a factor of $2kn - 1$ increase in runtime.*

Proof. Let $\mathbf{x} = (\mathbf{e}, \mathbf{s}) \in \{-\eta, \ldots, \eta\}^{2kn}$, $\mathbf{x}'$ be the guess given by the greedy solver. Assume w.l.o.g. that the zero scores $s_{j,0}$ for $j \in \{1, \ldots, 2n\}$ fulfill $s_{j,0} \leq s_{j+1,0}$ (otherwise, sort the key coefficients by zero scores), and let r be the number of recovered coefficients, i.e., $x_j = x'_j$ for $j \in \{1, \ldots, r\}$.

For every $r' \in \{1, \ldots, n\}$, the adversary assumes that r' coefficients have been recovered, assigns the most likely value to the first r' coefficients, and runs $\mathcal{A}$ with the best guess and these r' recovered coefficients. Now $\mathcal{A}$ outputs either the correct key or fail. But by premise, $\mathcal{A}$ recovers the secret key as soon as $r' = r$, which happens after at most $2kn - 1$ iterations.

Using the recovered coefficients and the best guess for the remaining coefficients, we may now directly apply the technique of [12]; see Section 2.5 for a summary. Note that the greedy solver (or the BP) do not have to be run again. In practice, a plausible interval for number of recovered coefficients can be estimated before carrying out the attack. Moreover, if $r' \leq r$, $\mathcal{A}$ may already recover the secret key. Thus, we have to assume that the adversary knows r.

Limitations. The greedy solver in general requires more hints than the BP-based solver. This does not come as a surprise, as the greedy solver computes an approximation to the probabilities computed during the BP. However, for a large number of hints, the BP becomes numerically infeasible, while the greedy solver can still recover the secret key.

The limitation of outliers in $\mathcal{D}$ is shared with the BP. If the true value of $\langle \mathbf{v}, \mathbf{x} \rangle$ is very unlikely with respect to $\mathcal{D}$, the greedy solver is more likely to select an action that worsens the distance to the true key. In the example of noise term leakage, artificially increasing the probabilities of values with positive but small probabilities leads to better results for the greedy solver as well.

5 Applications and Evaluation

Several previous attacks may be improved by modeling the side-channel information the adversary obtains as distribution hints and employing our solver. We first model several types of noise term leakage in ML-KEM and compare our solvers against the leaky LWE framework. Thereby, we improve upon a recent attack [27] on the subtraction during ML-KEM's decapsulation. We also describe how to make use of leakage on single coefficients of $\langle \mathbf{u}, \mathbf{s} \rangle$ which could improve upon previous attacks on the inverse NTT or lead to new attacks. Subsequently, we evaluate our solvers on artificially generated perfect and approximate hints.

Implementation and experimental setup. To simulate ML-KEM, we used PQClean [17]. The solvers are written in Rust (available as Python modules). All results for our solvers and the solvers of [12,27] were obtained as mean over 5 runs per setting. Let BIKZ denote the estimated smallest β such that BKZ-β recovers the secret key. To compare to [4], we used a single run of the estimator per setting and recorded the computational hardness after every 100 hints. This is possible because the [4] integrates hints sequentially; in addition, the variation in the resulting BIKZ is much smaller. We only provide comparisons for the selected setting because merely estimating the computational hardness using [4] requires far more time than fully recovering the secret key using our framework. This is in large parts because lattice-based frameworks are not parallelized.

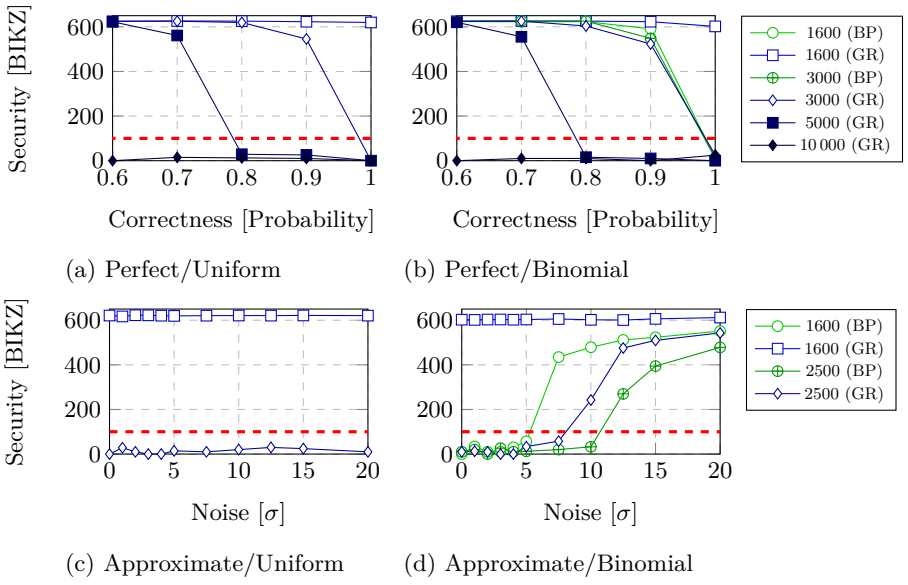
For the BP, we define an iteration to be passing messages from variable nodes to factor nodes and vice versa. For the greedy solver, we define an iteration as the number of updates until the number of applied actions is at $2kn$ again. To keep runtimes reasonable, we abort the solvers if BIKZ have not improved over 10 iterations. We also abort if more than half of the coefficients are correct (in that case the public key equation (2) allows recovering the remaining kn coefficients) or if the BIKZ is below 100 (shown by a dashed red line in the figures). Aborting at ≤ 100 BIKZ leads to a slight bias in the mean BIKZ to the upper end. Further, we rounded approximate hints to the nearest integer for our solvers; improving upon our implementation in this regard could further improve our results.

It should be noted that there usually is an interval of number of hints for which our solvers can in some cases fully solve for the secret key while it completely fails in other cases. Thus, whenever it does not impact readability, we additionally plot the area between minimum and maximum.

5.1 Perfect and Approximate Hints

To evaluate the performance on (erroneous) perfect and approximate hints, see Figure 3, we sample either uniformly random (over $\{-\lceil q/2 \rceil, \ldots, \lceil q/2 \rceil\}$) or binomially ($\eta = 5$) distributed coefficients. For perfect hints, we additionally simulate erroneous hints. This means that we sample a hint $\langle \mathbf{v}, \mathbf{x} \rangle = b$, but with probability p, we replace b by a uniformly random sampled value. The BP performs worse than the greedy solver on erroneous perfect hints and better on perfect hints. This seems to be caused by numerical issues. The occurrence of many uniform distributions seem to be particularly devastating to the BP, and the incorrectness of hints causes exactly this. For error-free perfect hints, our solvers require more hints than [4] but also greatly reduce the runtime ([18] can currently only handle hints on $\mathbf{s}$). For modular hints see Appendix B.

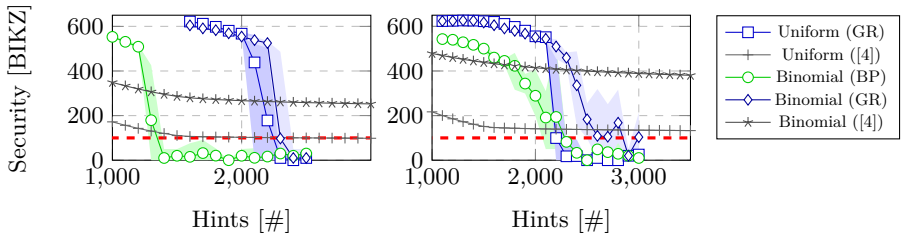
Figure 4 shows the comparison to [4] for $\sigma = 1.0$. In case of uniform coefficients with small σ, the greedy solver requires more hints to reach $\beta \approx 100$ but can then quickly fully recover the secret key, while [4] barely diminishes further. Furthermore, our solvers scale better with increasing σ; for example, with $\sigma = 10$, we can fully recover the secret key with 2300 hints while [4] still requires $\beta \approx 140$. For approximate hints with coefficients sampled from $\mathbf{B}_\eta$, our framework can recover the secret key in practice with fewer hints than [4] requires in order to reach $\beta \approx 300$. We thus expect our method to be favorable in the case of approximate hints in most real-world scenarios.



(a) Perfect/Uniform (b) Perfect/Binomial

(c) Approximate/Uniform (d) Approximate/Binomial

Fig. 3: Evaluation for perfect and approximate hints. Coefficients are either uniform over $\left\{\lfloor \frac{-q}{2} \rfloor, \ldots, \lfloor \frac{q}{2} \rfloor\right\}$ or sampled from $\mathbf{B}_5$.



(a) Noise $\sigma = 1$ (b) Noise $\sigma = 10$

Fig. 4: Comparison to [4]: Remaining computation hardness in BIKZ per number of applied hints for approximate hints with $\sigma \in \{1, 10\}$. Coefficients are either uniform over $\left\{\lfloor \frac{-q}{2} \rfloor, \ldots, \lfloor \frac{q}{2} \rfloor\right\}$ or sampled from $\mathbf{B}_5$.

5.2 Noise Term Leakage

The hints defined in Section 3.2 naturally occur in various attacks on LWE-based KEMs. These schemes rely on recovering the message from a noisy polynomial, and the noise term depends on the secret key and on both ciphertext components. Given the noise term and the ciphertext, the secret key can be obtained by solving a system of affine linear equations with small coefficients. Any kind of probabilistic information on a coefficient of the noise term can be expressed using our definition of hints. In this section, we analyze noise term leakage, compare against previous specialized solvers, and improve upon known attacks.

Recall from Section 2.2 that the message in Kyber is recovered in the decryption from

$$m' = v - \langle \mathbf{u}, \mathbf{s} \rangle = m + \mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top (\mathbf{e}_1 + \Delta \mathbf{u}) + e_2 + \Delta v \tag{20}$$

and may be written as $b_i = \langle \mathbf{a}_i, \mathbf{x} \rangle + c_i$, where $\mathbf{x} \in \mathbb{Z}^{2kn}$ is the flattened key and $\mathbf{a}_i \in \mathbb{Z}^{2kn}, c_i \in \mathbb{Z}$ are known to the adversary. Given information (in the form of a probability distribution) on any coefficient of the noise term b_i, we may thus derive a distribution hint. When doing so, we should always consider the prior distribution of b_i which we denote as P_{prior}.

Exploiting decryption failures. Our proof regarding the relation between BP and greedy solver seems contradictory to the results of [27]. In their work, the authors claim that their greedy solver outperforms BP-based solvers. Thus, we started by comparing the greedy solver of [27] against the back-then state-of-the-art BP-based solver of [12]. We do not yet directly compare against our solver because the attack of [27] outputs inequalities that already suffer from information loss – for a fair comparison we use the same inequalities as in [27]. In the next section, we simulate comparable leakage and show that directly working with distribution hints is advantageous (see Figure 7).

The results of our comparison are shown in Figure 5. It can be seen that the BP-based solver of [12] outperforms the greedy solver of [27] in terms of the required number of inequalities. We thereby disprove the claim in [27] that their greedy solver outperforms BP solvers by a factor of two. It should be noted that the greedy solver can be improved by several techniques. However, most of these improvements should translate to the BP of [12] as well [21] . Given Theorem 3, we argue that the greedy solver's disadvantage in this regard is inherent. It should be noted that deriving distribution hints instead of inequalities and applying our solvers greatly reduces the number of hints needed to recover the secret.

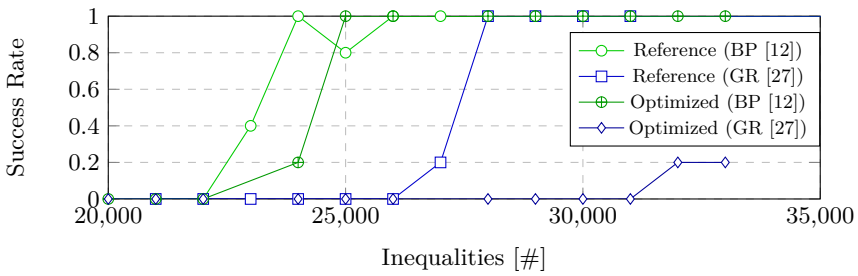


Fig. 5: Success rate per number of inequalities obtained in [27] for the greedy approach of [27] and the BP of [12]. The figure shows two settings of [27]: Targeting the reference and an optimized implementation of ML-KEM respectively.

Value leakage and comparison to [4]. We now first evaluate the performance of our solver in the case of value leakage, i.e., assuming that an adversary obtains the value of $b_i + \mathcal{N}(0, \sigma)$ (see (20)). This can be modeled as approximate hint[8] and both of our solvers and [4] may be used.

We simulate this type of leakage for $\sigma = 1.0$ and compare both our solvers against [4]. Note that running the lightweight version of [4] for estimation in many of our experiments took longer than fully recovering the secret key using our solvers. Also, in practice, an adversary would usually obtain HW leakage, which cannot be targeted by [4] at all (see next section).

The results are shown in Figure 6: The framework of [4] has a clear advantage for a small number of hints, but the reduction in security gradually declines with an increased number of hints. Our framework achieves better results starting from about 1300 hints. For an adversary that can for some reasons only obtain a very small number of hints, but has exceptionally large computational resources, [4] could be advantageous. However, we assume that recovering a few more hints is in practice almost always easier than obtaining computational resources far exceeding a public attack (i.e., running BKZ with $\beta \approx 300$). In

[8] But the equation the hints originate from is affine linear.

addition, in practice, the adversary will obtain HW leakage in almost all cases, which cannot be modeled using lattice-based frameworks.

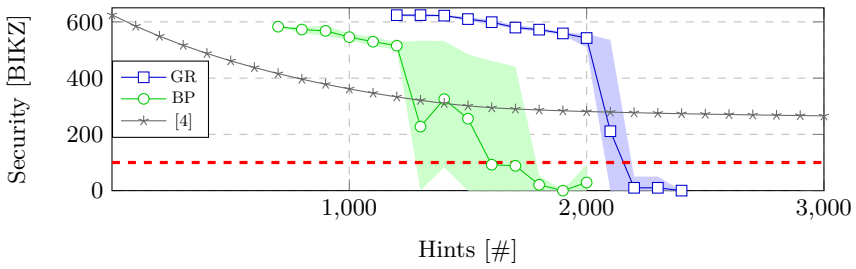


Fig. 6: Comparison to [4]: mean computation hardness in BIKZ per number of applied hints for noise term value leakage for $\sigma = 1$.

HW leakage and comparison to [27]. The attack of [27] derives inequalities, but, in fact, targets the subtraction of $v - \langle \mathbf{u}, \mathbf{s} \rangle$ (see Section 2.5) for ciphertexts where the message m is zero. Thereby, the attack obtains the HW of b_i directly. The authors of [27] derive inequalities from the HWs distributions. Instead, we may also directly express the information as distribution hints, which allows recovering the key with far fewer ciphertexts/traces.

If we obtain the Hamming weight of b_i, for any value w we get that the posterior distribution $P_{\text{post}}(w)$ is equal to

$$P(\langle \mathbf{a}_i, \mathbf{x} \rangle = w \mid \text{HW}(\langle \mathbf{a}_i, \mathbf{x} \rangle + c_i) = h) \tag{21}$$

$$\propto P(\text{HW}(\langle \mathbf{a}_i, \mathbf{x} \rangle + c_i) = h \mid \langle \mathbf{a}_i, \mathbf{x} \rangle = w) P_{\text{prior}}(\langle \mathbf{a}_i, \mathbf{x} \rangle = w) \tag{22}$$

and thus

$$P_{\text{post}}(w) = \begin{cases} P_{\text{prior}}(\langle \mathbf{a}_i, \mathbf{x} \rangle = w) & \text{if } \text{HW}(w + c_i) = h, \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

Similarly, we get that, if the Hamming weight of b_i follows a distribution $\mathcal{D}_{\text{HW}}$,

$$P_{\text{post}}(w) \propto P_{\mathcal{D}_{\text{HW}}}(w + c_i) P_{\text{prior}}(\langle \mathbf{a}_i, \mathbf{x} \rangle = w). \tag{24}$$

We did not perform any ciphertext filtering to reduce the size of c_i; similar to the results in previous work on decryption failure inequalities, we expect the results to be vastly better with ciphertext filtering in place.

Figure 7 shows the results of applying our solvers to HW leakage for different noise levels. For full key recovery, we require more than two times fewer ciphertexts with corresponding traces compared to [27].

Note that for the BP to work, we need to convolute the prior with a uniform distribution. Thereby, we prevent values with a low probability to become numerically zero in the factor node computations. See Appendix A for an evaluation for interval sizes for the uniform distribution.

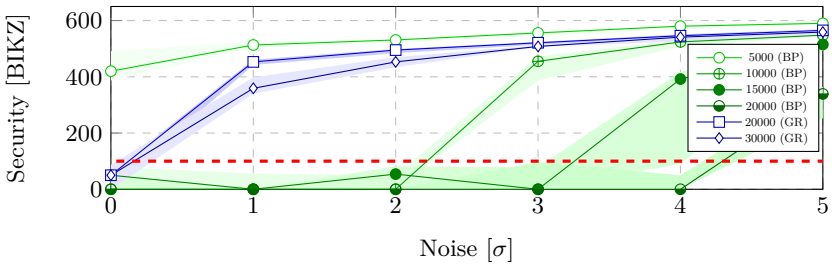


Fig. 7: Remaining computational hardness after applying our solver given HW leakage on the noise term in BIKZ per standard deviation.

Leakage on $\langle \mathbf{u}, \mathbf{s} \rangle$. Another location that may leak information on the noise term is the computation of $\langle \mathbf{u}, \mathbf{s} \rangle$. Previous attacks [24,11] that target the inverse NTT may recover such leakage. These attacks use BP to combine leakage in different layers of the NTT. It is well understood that fully recovering coefficients of the first layer allows deriving a lattice problem from which $\mathbf{s}$ can be recovered [11]. Using our method, fully recovering coefficients from the last layer gives another angle of attack. In fact, this results in value leakage on the noise term, and the evalaution in Section 5.2 applies.

An adversary could also target the summation of the inner product. In this case, they could obtain HW or value leakage on $\langle \mathbf{u}, \mathbf{s} \rangle$. In the case of HW leakage, the distribution for a hint can be computed by considering that

$$P_{\text{post}}(w) \propto P_{\mathcal{D}_{\text{HW}}}(v_i - w - c_i) P_{\text{prior}}(\langle \mathbf{a}_i, \mathbf{x} \rangle = w). \tag{25}$$

for any value w. For our evaluation, we only took coefficients into account for which both values of $c_i = (\Delta v + e_2)_i$ and v_i are small. Thereby, we greatly decrease the value range for $\langle \mathbf{u}, \mathbf{s} \rangle_i = v_i - w - c_i$, resulting in (simulated) measurements with a far higher information content. Figure 8 shows the results for leakage on $\langle \mathbf{u}, \mathbf{s} \rangle$ for $c_i = v_i = 0$.

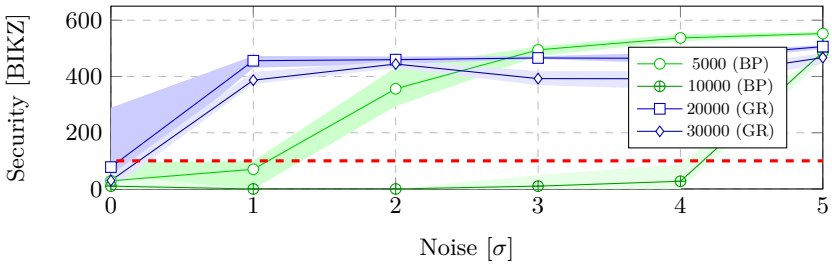


Fig. 8: Remaining computational hardness after applying our solver given HW leakage on $\langle \mathbf{u}, \mathbf{s} \rangle$ in BIKZ per standard deviation.

5.3 Comparison of Solvers

The greedy solver has several advantages: it is several orders of magnitudes faster, it overcomes the limitation of large coefficients, and it is less impacted by numerical limitations. The latter two cases cause the greedy solver be able to handle several types of information that the BP cannot solve. However, the BP requires fewer hints in various settings.

Runtime. The BP usually finishes within a few hours on a single core and scales very well with the number of cores (see Appendix C). Thus, in practice, for non-corner cases, the key can often be recovered in a few minutes on widely-available hardware. However, in corner cases, we also observed runtimes of a day on 20 cores. This is unlikely to prevent determined adversaries, especially as it is highly parallelizable; though, it may cause difficulties in designing attacks. The greedy solver is usually done in a few minutes even on very few cores.

Lattice-based solvers. Both solvers require more hints in noise-/error-free settings than lattice-based solvers. However, the drastically reduced runtime could still cause them to be favorable for a real-world attacker when compared to [4], and [18] currently only applies to perfect and modular hints on $\mathbf{s}$. In the case of approximate hints, the BP outperforms [4] but cannot handle large coefficients. As soon as information becomes erroneous, non-uniformly noisy, or information is not obtained in the value leakage model, only our solver can be used.

6 Conclusion

In this work, we provide a framework to handle generic probabilistic side-channel information in lattice-based cryptography. We show that distribution hints are a generalization of almost all of the hints previously defined for lattice-based frameworks. Further, we prove that solvers used in practice are special cases of our algorithms and explained the relation between these algorithms. Our evaluations show that several practical attacks as well as artificial settings benefit from the usage of distribution hints and our solving methods. In short: Whenever information is erroneous or noisy, our framework has an advantage or are often even the only viable option.

Future work and open problems. The BP-based solver is limited in two regards: Large coefficients and numeric errors often prevent recovery of the secret key. The first one is intrinsic and unlikely to be circumvented: Large coefficients lead to costly computations. On the other hand, the second limitation is of numerical nature and can most likely be circumvented with better implementation techniques. We partially dealt with this issue by convoluting with a uniform distribution – while this works, it is most likely suboptimal. Finding techniques to circumvent these issues more elegantly is yet an open question.

Further, we cannot yet give any precise relation between information content and convergence of our solvers. In previous work [12], some evaluation in this

direction has been provided for decryption failure information. But the relation between the information content and convergence is not yet fully understood. The convergence of cyclic BP seems to be particularly hard to analyze, and, in our case, the BP graph is additionally fully connected.

Another open question is whether the connection between greedy algorithms and BP holds more generally. Our greedy algorithm can be seen as a "collapsed" BP. In our opinion, it is an interesting question under which circumstances we can derive a BP instantiation from a greedy algorithm.

References

1. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.
2. Robert Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
3. Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):334–359, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8977`.
4. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
5. Dana Dachman-Soled, Huijing Gong, Tom Hanson, and Hunter Kippen. Revisiting security estimation for LWE with hints from a geometric perspective. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 748–781, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Heidelberg, Germany.
6. Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. Higher-order masked ciphertext comparison for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(2):115–139, 2022.
7. Jeroen Delvaux. Roulette: A diverse family of feasible fault attacks on masked kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):637–660, 2022.
8. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
9. Michael Fahr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray A. Perlner, Arkady Yerukhimovich, and Daniel Apon. When frodo flips: End-to-end key recovery on FrodoKEM via rowhammer. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 979–993, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.
10. Robert G. Gallager. Low-density parity-check codes. *IRE Trans. Inf. Theory*, 8(1):21–28, 1962.
11. Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):88–113, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/9061`.

12. Julius Hermelink, Erik Mårtensson, Simona Samardjiska, Peter Pessl, and Gabi Dreo Rodosek. Belief propagation meets lattice reduction: Security estimates for error-tolerant key recovery from decryption errors. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):287–317, 2023.
13. Julius Hermelink, Kai-Chun Ning, and Emanuele Strieder. The insecurity of masked comparisons: Scas on ml-kem's fo-transform. *IACR Cryptol. ePrint Arch.*, page 60, 2024.
14. Julius Hermelink, Kai-Chun Ning, and Emanuele Strieder. The insecurity of masked comparisons: SCAs on ML-KEM's FO-transform. Cryptology ePrint Archive, Paper 2024/060, 2024. `https://eprint.iacr.org/2024/060`.
15. Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on kyber. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 311–334. Springer, 2021.
16. Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):243–268, 2020. `https://tches.iacr.org/index.php/TCHES/article/view/8590`.
17. Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. Improving software quality in cryptography standardization projects. In *IEEE European Symposium on Security and Privacy, EuroS&P 2022 - Workshops, Genoa, Italy, June 6-10, 2022*, pages 19–30, Los Alamitos, CA, USA, 2022. IEEE Computer Society.
18. Alexander May and Julian Nowakowski. Too many hints - when LLL breaks LWE. In *Advances in Cryptology – ASIACRYPT 2023, Part IV*, Lecture Notes in Computer Science, pages 106–137. Springer, Heidelberg, Germany, December 7–11, 2023.
19. National Institute of Standards and Technology. Module-lattice-based digital signature standard. Technical report, Department of Commerce, Washington, D.C., 2023. Federal Information Processing Standards Publication (FIPS) NIST FIPS 204 ipd. `https://doi.org/10.6028/NIST.FIPS.204.ipd`.
20. National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical report, Department of Commerce, Washington, D.C., 2023. Federal Information Processing Standards Publication (FIPS) NIST FIPS 203 ipd. `https://doi.org/10.6028/NIST.FIPS.203.ipd`.
21. Thales Paiva and Julius Hermelink. Personal conversation, 2024.
22. Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149, Santiago, Chile, October 2–4, 2019. Springer, Heidelberg, Germany.
23. Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):37–60, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8787`.
24. Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529

of *Lecture Notes in Computer Science*, pages 513–533, Taipei, Taiwan, September 25–28, 2017. Springer, Heidelberg, Germany.

25. Zehua Qiao, Yuejun Liu, Yongbin Zhou, Mingyao Shao, and Shuo Sun. When NTT meets SIS: Efficient side-channel attacks on dilithium and kyber. Cryptology ePrint Archive, Paper 2023/1866, 2023. `https://eprint.iacr.org/2023/1866`.
26. Gokulnath Rajendran, Prasanna Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. Pushing the limits of generic side-channel attacks on LWE-based KEMs - parallel PC oracle attacks on kyber KEM and beyond. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):418–446, 2023.
27. Prasanna Ravi, Thales Paiva, Dirmanto Jap, Jan-Pieter D'Anvers, and Shivam Bhasin. Defeating low-cost countermeasures against side-channel attacks in lattice-based encryption A case study on crystals-kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):795–818, 2024.
28. Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, 2020. `https://tches.iacr.org/index.php/TCHES/article/view/8592`.
29. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.
30. Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David F. Oswald, Wang Yao, and Zhiming Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Trans. Computers*, 71(9):2163–2176, 2022.

A Numerical Issues

In Figure 9, we visualize the numerical problems described in Section 4.1. For a large number of hints, the BP-based solver cannot solve for the secret key anymore (compare with Figure 7) in low noise levels, but larger amounts of noise improve convergence.

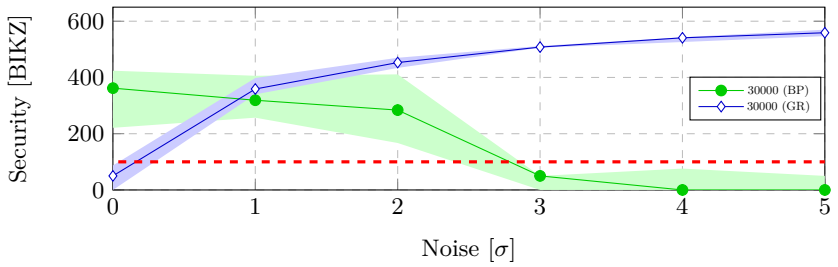


Fig. 9: Remaining computational hardness after applying our solver given HW leakage on the noise term in BIKZ per standard deviation. The BP graph for 30000 hints shows the numerical problems that can occur with too many hints.

For HW leakage, we have to convolute the hint distributions with a uniform distribution on an interval of length $2c$ to avoid non-zero but small probabilities to become numerically zero during the BP. For the BP, this phenomenon worsens with increasing number of hints and low noise levels. Figure 10 shows our evaluations for several sizes of intervals.

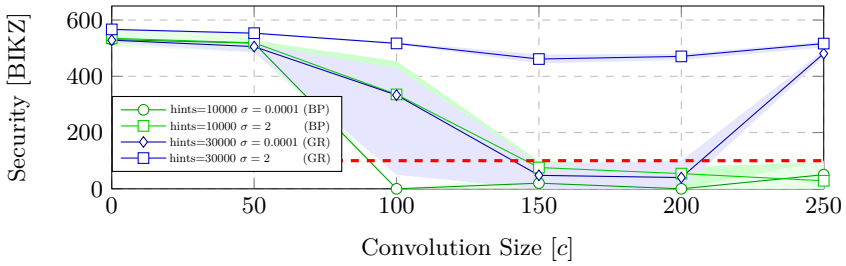
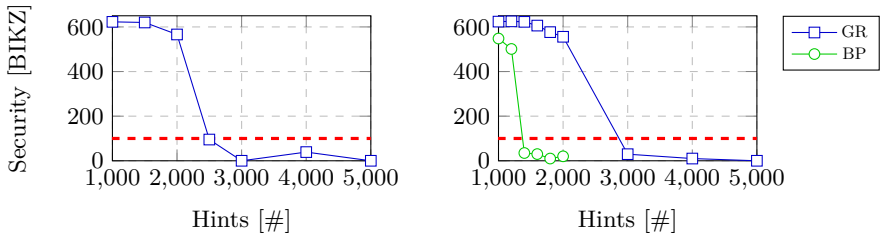


Fig. 10: Applying BP and greedy solvers to HW noise term leakage for different uniform distributions on intervals of size c.

B Modular Hints

We did not manage to recover the secret from modular hints with uniform coefficients. For small coefficients, we require more hints than lattice-based frame-

works. Figure 11 shows our evaluations for binomially sampled coefficients ($\eta = 5$) and coeffcients sampled uniformly random over $\{-20, \ldots, 20\}$;



(a) Uniform (b) Binomial

Fig. 11: Evaluation for modular hints.

C Runtime

In Table 1, we report on the runtimes for solving perfect and approximate hints (with $\sigma = 10$ and binomial coefficients) on a single core and on 40 cores on two `Intel Xeon Gold 6230`. The results are averaged over 5 runs each. Note that we did not ensure ideal benchmarking conditions. Thus, these numbers are only rough estimates.

For the BP, we suspect memory throughput to be the limiting factor and Hyperthreading seems to be disadvantageous if many hints are integrated. We in some cases observed threads going into uninteruptable sleep. We cannot pinpoint the exact reason for this behavior but suspect it to be related to limited memory throughput. This also occurs in the implementations of [15,12]. Both of these share a common core, while our implementation is independent but reuses the FFT-based strategy that was proposed by [23] and has also been used in [15,12].

Table 1: Mininmum, maximum, and average runtimes on 1 and 40 cores for approximate and perfect hints in minutes (min./max./avg.).

Setting	Approx. (2000)	Approx. (5000)	Perfect (2000)	Perfect (5000)
GR (1 core)	7.08/14.11/9.28	1.87/2.05/1.97	1.52/2.63/2.10	0.47/0.60/0.51
GR (40 cores)	1.00/1.98/1.43	0.26/0.41/0.32	0.67/1.50/1.07	0.18/0.32/0.23
BP (1 core)	– – –	– – –	308.51/368.18/344.22	318.28/467.10/348.71
BP (40 cores)	38.01/55.01/46.73	8.48/27.36/18.17	9.18/10.98/9.95	9.94/10.31/10.11