IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 4, pp. 88–113. DOI:10.46586/tches.v2021.i4.88-113

Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber

Mike Hamburg[1], Julius Hermelink[2], Robert Primas[3], Simona Samardjiska[4], Thomas Schamberger[5], Silvan Streit[6], Emanuele Strieder[7] and Christine van Vredendaal[8]

[1] Rambus Labs, San Jose, USA mhamburg@rambus.com
[2] Universität der Bundeswehr München, Munich, Germany julius.hermelink@unibw.de
[3] Graz University of Technology, Graz, Austria robert.primas@iaik.tugraz.at
[4] Radboud University, Nijmegen, The Netherlands simonas@cs.ru.nl
[5] Technical University of Munich (TUM), Munich, Germany t.schamberger@tum.de
[6] Fraunhofer Institute AISEC, Garching near Munich, Germany silvan.streit@aisec.fraunhofer.de
[7] Fraunhofer Institute AISEC, Garching near Munich, Germany emanuele.strieder@aisec.fraunhofer.de
[8] NXP Semiconductors, Eindhoven, The Netherlands christine.cloostermans@nxp.com

Abstract.
Single-trace attacks are a considerable threat to implementations of classic public-key schemes, and their implications on newer lattice-based schemes are still not well understood. Two recent works have presented successful single-trace attacks targeting the Number Theoretic Transform (NTT), which is at the heart of many lattice-based schemes. However, these attacks either require a quite powerful side-channel adversary or are restricted to specific scenarios such as the encryption of ephemeral secrets. It is still an open question if such attacks can be performed by simpler adversaries while targeting more common public-key scenarios.
In this paper, we answer this question positively. First, we present a method for crafting ring/module-LWE ciphertexts that result in sparse polynomials at the input of inverse NTT computations, independent of the used private key. We then demonstrate how this sparseness can be incorporated into a side-channel attack, thereby significantly improving noise resistance of the attack compared to previous works. The effectiveness of our attack is shown on the use-case of CCA2 secure Kyber k-module-LWE, where $k \in \{2, 3, 4\}$. Our k-trace attack on the long-term secret can handle noise up to a $\sigma \leq 1.2$ in the noisy Hamming weight leakage model, also for masked implementations. A $2k$-trace variant for Kyber1024 even allows noise $\sigma \leq 2.2$ also in the masked case, with more traces allowing us to recover keys up to $\sigma \leq 2.7$. Single-trace attack variants have a noise tolerance depending on the Kyber parameter set, ranging from $\sigma \leq 0.5$ to $\sigma \leq 0.7$. As a comparison, similar previous attacks in the masked setting were only successful with $\sigma \leq 0.5$.

Keywords: Kyber, NTT, belief propagation, side-channel attack, CCA, BKZ

1 Introduction

Current public-key cryptographic schemes are based on the premise that the mathematical problems underlying them are hard to solve for the chosen parameters. With the advent of a quantum computer however, these classical hard problems will be efficiently solvable by applying Shor's algorithm [Sho94]. As a result, there is rising interest in post-quantum cryptography (PQC) algorithms, which are based on mathematical problems conjectured

Received: 2021-04-15 Accepted: 2021-06-15 Published: 2021-08-11

to resist quantum attack. To facilitate the standardization of such algorithms, the National Institute of Standards and Technology (NIST) in 2017 put out a call to submit PQC candidates [Nat] with the aim to be standardized. Currently, there are 15 candidates in Round 3 of the evaluation, a few of which are expected to be standardized over the next years.

One promising candidate is Kyber [BDK+18]. It is part of the class of lattice-based cryptography, and in particular a module-Learning With Errors (module-LWE) scheme. Kyber implementations rely on the fact that its ring, the cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, allows for polynomial multiplication to be performed by the Number Theoretic Transform (NTT), a discrete variant of a Fast Fourier Transform (FFT).

The threat of *side-channel attacks*, introduced by [KJJ99], on cryptographic implementations is widely acknowledged. Many types of side-channel attacks are known, ranging from exploiting the timing to Electromagnetic (EM) radiation, from exploiting visible key bits to distinguishing subtle differences using machine learning, and from requiring millions of scope traces down to only one. Because post-quantum cryptography is a relatively new field, the landscape of threats has not been mapped as thoroughly.

Related Work For lattice-based constructions side-channel work so far is mostly focused on applying *masking* to implementations, which is aimed at increasing the difficulty of various degrees of differential attacks [RRVV15, RdCR+16, RRdC+16, OSPG18, BPO+20].

On the attack side, observations on using the properties of an NTT to recover ring-LWE keys from partial key recovery were made in [BBPS19, DGKS20]. In [RBRC20] an attack for the message recovery in unprotected NIST PQC LWE/LWR schemes is presented. They extend their attack to a masked implementation, but can only perform message recovery. Very recently, a 16-trace attack against a first-order masked implementation of IND-CCA secure Saber KEM was presented in [NDGJ21]. It describes how to both recover the session key and the long-term secret key by applying deep learning techniques. This removes the limitations of [RBRC20]. Some attacks can even compromise a message or private key using only a single trace. In particular, [PPM17] and [PP19] recover data passed through an NTT by templating the multiplications or other intermediate values within the NTT. An extensive description of these attacks, including the used *belief propagation* technique, is presented in Section 3. These single-trace attacks are still limited in that they either require extensive profiling efforts or in that they are only applicable in specific scenarios like the encryption of ephemeral keys. The question of whether a variant of these attacks could be used for key recovery in public-key decryption while requiring only modest profiling efforts was so far unanswered.

Contributions In this work we show that a sparse Chosen Ciphertext Attack (CCA), combined with belief propagation can more efficiently recover (CCA2-secure masked) Kyber keys from side-channel information on the NTT computations. The attack requires k traces on the inverse NTT step of Kyber decryption, where $k \in \{2, 3, 4\}$ is the module dimension, for a noise tolerance $\sigma \leq 1.2$ in the Hamming weight leakage. More specifically:

- We present a novel sparse-vector CCA strategy, which leads to full long-term key recovery. This strategy improves the effectiveness of belief propagation of [PPM17, PP19] by allowing larger noise levels in the traces without reducing the recovery rate, also in the masked case.
- We demonstrate how to choose the number of traces in our attack from 1 to k to increase the noise tolerance from $\sigma \leq 0.5 - 0.7$ up to $\sigma \leq 1.2$ in the Hamming weight leakage. Our simpler sparse vector generation based on the NTT structure can still handle noise up to $\sigma \leq 0.9$ in a k-trace attack. For Kyber1024 we are even able to handle noise up to $\sigma \leq 2.2$ in an $2k$-trace attack. With repeating failed runs we can

further increase the noise up to $\sigma \leq 1.4$ for Kyber512 and Kyber768, and $\sigma \leq 2.7$ for Kyber1024.

- We show that the attack is applicable to masked implementations of Kyber, when the masking is applied to the secret key. This is a common masking strategy and used as a part of the masking in, for instance, [RRdC+16, OSPG18].
- We provide an implementation of the attack, to showcase its effectiveness. Despite its reliance on chosen ciphertexts, our implementation shows that it can be applied to CCA2-secure constructions.

Open Source The open-source implementation of our attack can be found at https://github.com/BayesianSCA/k-trace-CCA.

Outline The remainder of this paper is organized as follows. In Section 2 we review the necessary preliminaries on Kyber and the NTT. In Section 3 we explain the belief propagation SCA presented in [PPM17, PP19], as well as its limitations. In Section 4 we put forth our novel sparse-vector NTT attack which improves on that attack, the effectiveness of which is shown in Section 5. In Section 6 we discuss countermeasures and possibilities of extending out attack to other lattice-based schemes. We conclude our paper in Section 7.

2 Preliminaries

In this section we introduce the module Learning With Errors problem, go into some detail of Kyber, and introduce the Number Theoretic Transform.

2.1 Learning With Errors

The Learning With Errors (LWE) problem [Reg05] and its instantiation over rings [LPR10] or modules are the basis of multiple NIST PQC candidates.

Let $\mathbb{Z}_q$ be the ring of integers modulo q and for given degree n, define $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ as the polynomial ring of polynomials modulo $x^n + 1$. Let also β_η denote the centered binomial distribution with parameter η and $\mathcal{U}$ the uniform distribution over $\mathbb{Z}_q$. A module-LWE distribution now consists of tuples $(\mathbf{a}, b = \mathbf{a}^T\mathbf{s} + e) \in \mathbb{Z}_q^k \times \mathbb{Z}_q$ (resp. $\mathcal{R}_q^k \times \mathcal{R}_q$), where coefficients of $\mathbf{s}$ are drawn from β_η once, and for each sample e is freshly drawn from β_η and the coefficients of $\mathbf{a}$ from $\mathcal{U}$. The module-LWE based schemes rely on the hardness of recovering $\mathbf{s}$ from several of such tuples.

2.2 Kyber

Kyber [BDK+18] is a Key Encapsulation Mechanism (KEM) submitted to the NIST standardization process. It is among the 7 finalists of the 15 schemes in Round 3 [Nat]. Its security is based on the module-LWE problem. For the three parameter sets in the proposal, Kyber512, Kyber768, and Kyber1024, the parameters are all set to $n = 256$ and $q = 3329$. For most parameters $\eta = 2$ is used, except for Kyber512, where $\eta = 3$. The parameter sets differ in their module dimension $k = 2, 3$, and 4 respectively. Since our focus is on its NTT (see Section 2.3), a simplified version suffices, omitting details such as how coefficients are packed. Kyber also supports "90s" versions of each parameter set, which substitute AES and SHA2 for SHAKE and SHA3, but this distinction doesn't affect our attack because we target only the NTT. For further details on Kyber, we refer to [BDK+18]. Kyber's CCA2-KEM Key Generation, PKE- and CCA2-KEM-Encryption, and CCA2-KEM-Decryption are summarized in Algorithms 1, 2, 3 and 4.

Algorithm 1 Kyber-CCA2-KEM Key Generation (simplified)

Output: Public key pk, secret key sk

1: Choose uniform seeds ρ, σ, z
2: $R_q^{k\times k} \ni \hat{\mathbf{A}} \coloneqq \mathsf{Sample}_{\mathsf{U}}(\rho)$ ▷ Generate uniform $\hat{\mathbf{A}}$ in NTT domain
3: $R_q^k \ni \mathbf{s}, \mathbf{e} \coloneqq \mathsf{Sample}_{\mathsf{B}}(\sigma)$ ▷ Sample from binomial distribution
4: $\hat{\mathbf{s}} \coloneqq \mathsf{NTT}(\mathbf{s})$ ▷ NTT for efficient multiplication
5: $\hat{\mathbf{t}} \coloneqq \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \mathsf{NTT}(\mathbf{e})$ ▷ $\mathbf{t} \coloneqq \mathbf{As} + \mathbf{e}$
6: **return** $(pk \coloneqq (\hat{\mathbf{t}}, \rho), sk \coloneqq (\hat{\mathbf{s}}, pk, \mathsf{Hash}(pk), z))$

In these algorithms, and in the rest of this paper, the notation $a \cdot b$ means ordinary multiplication, whereas $a \circ b$ means "pairwise-pointwise" multiplication of polynomials, or vectors or matrices of polynomials, in the NTT domain. Compression is defined as $\mathsf{Compress}(u) := \lfloor u \cdot 2^d/q \rceil \mod 2^d$. This lossily compresses an element of $\mathbb{Z}_q$ to d bits, where d varies across the element being compressed and across Kyber instances.

The PKE-Encryption is shown in Algorithm 2. The seed τ used for the noise sampling is made explicit to allow the re-encryption required for the CCA2 transform in Algorithm 3. The ciphertext c consists of two compressed parts, where the second component c_2 contains m encoded as an element in $\mathcal{R}_q$. The decryption process (Algorithm 4) requires the recipient to recover this m from a noisy version.

Algorithm 2 Kyber-PKE Encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{t}}, \rho)$, message m, seed τ
Output: Ciphertext c

1: $\hat{\mathbf{A}} \in R_q^{k\times k} \coloneqq \mathsf{Sample}_{\mathsf{U}}(\rho)$ ▷ Regenerate uniform $\hat{\mathbf{A}}$
2: $\mathbf{r}, \mathbf{e}_1 \in R_q^k, e_2 \in R_q \coloneqq \mathsf{Sample}_{\mathsf{B}}(\tau)$ ▷ Sample noise $\mathbf{r}, \mathbf{e}_1, e_2$ (binomial)
3: $\mathbf{u} \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \mathsf{NTT}(\mathbf{r})) + \mathbf{e}_1$ ▷ $\mathbf{u} \coloneqq \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
4: $v \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \mathsf{NTT}(\mathbf{r})) + e_2 + \mathsf{Encode}(m)$ ▷ $v \coloneqq \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Encode}(m)$
5: $\mathbf{c}_1, c_2 \coloneqq \mathsf{Compress}(\mathbf{u}, v)$ ▷ Round off lower bits (lossy)
6: **return** $c \coloneqq (\mathbf{c}_1, c_2)$

Algorithm 3 Kyber-CCA2-KEM Encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{t}}, \rho)$
Output: Ciphertext c, shared key K

1: Choose uniform m
2: $(\bar{K}, \tau) \coloneqq \mathsf{Hash}(m||\mathsf{Hash}(pk))$ ▷ Embed pk into encryption
3: $c \coloneqq \mathsf{PKE.Enc}(pk, m, \tau)$ ▷ CPA encryption with fixed seed
4: $K \coloneqq \mathsf{KDF}(\bar{K}||\mathsf{Hash}(c))$ ▷ Derive shared key
5: **return** (c, K)

Kyber uses a variant of the Fujisaki-Okamoto transform [FO13] to build an IND-CCA2 secure key-encapsulation mechanism (KEM). This transform applies an additional re-encryption of the decrypted message (cf. Algorithm 4, l. 4), using the same randomness as used for the encryption of the received ciphertext. The decryption is only deemed valid if the re-computed ciphertext matches the received ciphertext. As our attack exploits leakage that occurs before the re-encryption, i.e. NTT^{-1} in line 2, the check does not mitigate our attack.

Algorithm 4 Kyber-CCA2-KEM Decryption (simplified)

Input: Secret key $sk = (\hat{\mathbf{s}}, pk, h, z)$, ciphertext $c = (\mathbf{c}_1, c_2)$
Output: Shared key K

1: $\mathbf{u}, v \coloneqq \mathsf{Decompress}(\mathbf{c}_1, c_2)$ ▷ Expand to full modulus
2: $m \coloneqq \mathsf{Decode}(v - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u})))$ ▷ $m \coloneqq \mathsf{Decode}(v - \mathbf{s}^T\mathbf{u})$
3: $(\bar{K}, \tau) \coloneqq \mathsf{Hash}(m||h)$ ▷ Regenerate seed for encryption
4: $c' \coloneqq \mathsf{PKE.Enc}(pk, m, \tau)$ ▷ Assert valid ciphertext
5: $K \coloneqq \mathsf{KDF}(\bar{K}||\mathsf{Hash}(c))$ if $c = c'$, ▷ Derive shared key
6: else $K \coloneqq \mathsf{KDF}(z||\mathsf{Hash}(c))$ ▷ Implicit rejection on failure
7: **return** K

2.3 Number Theoretic Transform

For lattice-based schemes using polynomial rings, the polynomial multiplication is the most computationally expensive step. The Number Theoretic Transform (NTT) is a technique that enables efficient computation of this multiplication.

The NTT is similar to the Discrete Fourier Transform (DFT), but instead of over the field of complex numbers, it operates over a prime field $\mathbb{Z}_q$. It can be seen as a mapping between the coefficient representation of a polynomial from $\mathcal{R}_q$ (the *normal domain*) to the evaluation of the polynomial at the n-th roots of unity (*the NTT domain*). This bijective mapping is typically referred to as forward transformation. The mapping from the NTT domain to the normal domain is referred to as backward transformation or inverse NTT. In the NTT domain, multiplication of polynomials can be achieved by point-wise multiplication, which is much cheaper than multiplication in the normal domain. Typically, one would perform the forward transformation, multiply the polynomials (point-wise) in the NTT domain, and go back using the backward transformation.

For $\mathcal{R}_q$ with a $2n$-th primitive root of unity ζ, the NTT transformation of an n-degree polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ is defined as:

$$\hat{f} = \mathsf{NTT}(f) = \sum_{i=0}^{n-1} \hat{f}_i x^i, \text{ where } \hat{f}_i = \sum_{j=0}^{n-1} f_j \zeta^{(2i+1)\cdot j}. \tag{1}$$

Similarly,

$$f = \mathsf{NTT}^{-1}(\hat{f}) = \sum_{i=0}^{n-1} f_i x^i, \text{ where } f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \zeta^{-i\cdot(2j+1)}. \tag{2}$$

The NTT transform (and its inverse) can be applied efficiently by using a chaining of $\log_2 n$ *butterflies*. It is a divide and conquer technique that splits the input in half in each step and solves two problems of size $n/2$. For $n = 2^k$ after k steps, the problems are of size 1 and can be trivially solved. The way the splitting is done is referred to as *decimation* and typically in practice, either the Cooley-Tukey [CT65] or the Gentleman-Sande [GS66] butterfly is used. The construction for a 8-coefficient NTT using the Cooley-Tukey butterfly with decimation in time is depicted in Figure 2.1 (cf. [CT65]), with the output being in bit-reversed order.

Schemes like Kyber and Dilithium [DKL+18] use an NTT-friendly ring. But in Kyber, only n-th primitive roots of unity exist, therefore the modulus polynomial $X^n + 1$ only factors into polynomials of degree 2. Hence, the last layer of the NTT is skipped (nearest neighbors) and in NTT domain multiplication is not purely pointwise, but multiplications of polynomials of degree one (*pairwise-pointwise*). That is, the Kyber ring is effectively $\mathbb{F}_{q^2}[y]/(y^{128}+1)$, where $\mathbb{F}_{q^2}$ is the field $\mathbb{Z}_q[x]/(x^2-\zeta)$. Also note that in Kyber, polynomials in NTT domain are always considered in bit-reversed order (cf. Figure 2.1). Therefore,

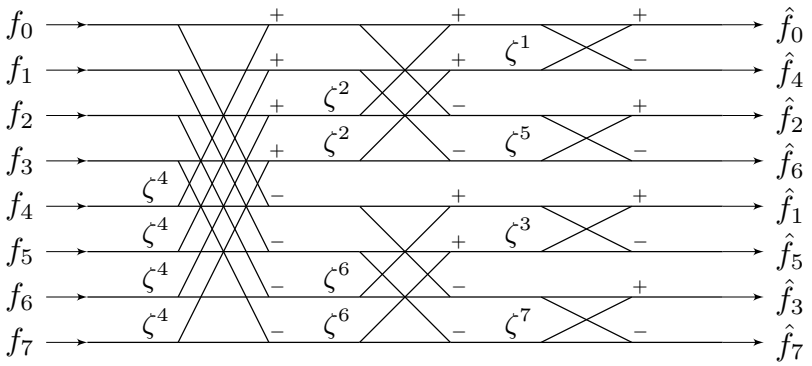


Figure 2.1: 8-coefficient Cooley-Tukey decimation in time NTT

in the following bit-reversal is implicitly expected in the NTT domain and indices for NTT-coefficients are noted in regular order.

It was recently shown that schemes like NTRU [ZCH+20], Saber [DKR+20], and NTRU Prime [BBC+20] can also benefit from the NTT by applying a transformation from their NTT-unfriendly-ring to a friendly one [ACC+21a, CHK+21].

2.4 Masked Implementations

Several previous works propose DPA-secured implementations of lattice-based schemes that use masking as their main protection mechanism. The first masking scheme for ring-LWE decryption was presented by Reparaz et al. [RRdC+16] and was later extended to a CCA2-secure version by Oder et al. [OSPG18]. An alternative countermeasure, more in line with classical blinding techniques, utilizes the additively homomorphic nature of ring-LWE and is presented in [RdCR+16].

For the purpose of this paper, we are only interested in the initial decryption of the ciphertext, containing the inverse NTT operation, as we do not exploit leakage of subsequent computation steps. Here, the application of classical masking is rather straightforward as the NTT itself is a linear operation. Hence, one can simply (1) split the secret key into two (or more) shares, (2) multiply the NTT-domain ciphertext by each share, (3) compute the inverse NTTs independently for each share, and (4) finally add the shares back again if needed [RRdC+16, OSPG18]. The same strategy is also applicable for module-LWE-based schemes like Kyber. We give a more detailed description of masked Kyber decryption in Section 5.1.

3 Soft-Analytical Side-Channel Attacks (SASCA)

In this section, we first give a generic description of Soft-Analytical Side-Channel Attacks (SASCA) and the Belief Propagation algorithm (BP), which are frequently used in profiled power analysis attacks. We base our descriptions of BP on MacKay [Mac03, Chapter 26] and on previous works using SASCA [VGS14, PP19]. We then outline previous works that study profiled power analysis attacks on (masked) NTT computations and point out their limitations. We do not discuss DPA attacks, as masking is generally considered to be an effective countermeasure against these kinds of attacks.

3.1 Belief Propagation

Profiled power analysis attacks in the single/few-trace setting often face the problem that measurements alone are not sufficient to determine the exact values of an analyzed computation. A common way to reduce the remaining guessing entropy is to build systems of equations which relate multiple intermediate values, and to solve them using SAT solvers or sophisticated brute force algorithms. In 2014, this idea was extended by Veyrat-Charvillon et al. [VGS14] proposing soft-analytical side-channel attacks (SASCA). This class of attacks treats the reduction of the remaining guessing entropy in combination with algorithm knowledge as a noisy decoding problem. Belief propagation is an inference algorithm which has proven to be very useful when decoding these kinds of problems. Knowledge about the algorithm is transformed into a so called factor-graph. This factor-graph models intermediate values of the algorithm as *variable nodes* and the relations or transformations between the intermediate values as *factor nodes*. Each variable node represents a local marginal probability distribution of the global joint distribution of all intermediate values. The variable nodes hold the initial probabilities - also called beliefs - which were gained via classical template matching. Variable and factor nodes then alternately exchange messages about their beliefs. This iterative process reduces intractable decoding solutions and could disclose the true values of each intermediate of the secret. A more thorough and formal explanation of BP is given in Appendix A.

The application of BP in side-channel analysis has proven to be very powerful [VGS14, PPM17, KPP20, PP19, GRO18]. However, BP only decodes the correct marginals if the factor graph has no cycles. If BP is used in cyclic graphs, we refer to it as loopy Belief Propagation. The quality of the solutions of loopy BP is inversely proportional to the length of the loops. Short loops can introduce overconfidence, which could lead to oscillations or incorrect solutions within local minima [PP19]. Therefore, it is beneficial to overcome short loops by clustering or remodeling of factor nodes which introduce these short loops. Additional attention has to be paid with respect to iteration number and break conditions in loopy BP.

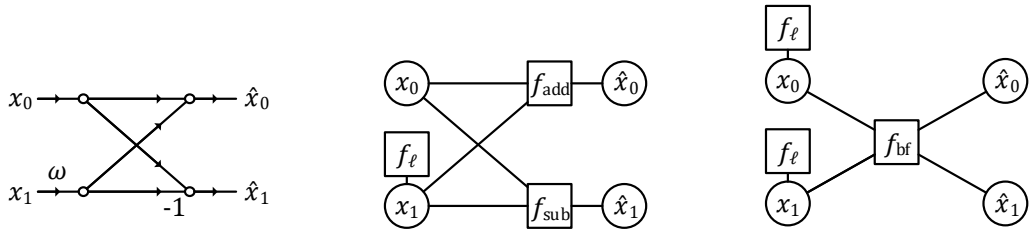
3.2 Prior Work

As shown in prior work [PPM17, PP19], attacks can recover sensitive NTT inputs after observing just a single trace. More concretely, they recover the inputs to the forward/inverse NTT during ring-LWE encryption/decryption respectively. While decryption is generally the more interesting attack target since it involves the usage of a long-term private key, single-trace attacks on the encryption are also plausible attack scenarios for KEMs where ephemeral secrets are being encrypted.

Fig. 3.1 illustrates some options to construct a factor graph for an NTT computation. Fig. 3.1a shows a single butterfly which is equivalent to a length-2 NTT and transforms an input polynomial with coefficients x_0, x_1 into the corresponding output polynomial in the NTT domain with coefficients $\hat{x}_0, \hat{x}_1$. Fig. 3.1b depicts the corresponding factor graph as constructed in [PPM17]. In this graph, variable nodes (intermediate values) and factor nodes (computations) are represented by circles and squares, respectively.

The factor nodes can be further split into two groups. The first group of factors f_ℓ model the observed side-channel information, i.e., the outcome of the template matching $f_\ell(i) = \Pr(x = i|\ell)$ where x is the matched intermediate and ℓ is the observed side-channel leakage. In [PPM17] the template matching was performed on the modular multiplication with ω (corresponding to the powers of ζ in the NTT, cf. Figure 2.1), and hence where they receive information on x_1. In this case, modeling the effects of multiplication on the beliefs simply corresponds to shuffling their probabilities since ω is publicly known.

The second group of factors, consisting of f_{add} and f_{sub}, model the deterministic relationships between the variable nodes as specified by the NTT. E.g., for the addition in



(a) Single butterfly

(b) Simple factor graph representation of a butterfly as used in [PPM17]

(c) The factor graph used in [PP19] and in our work.

Figure 3.1: Comparison of a single butterfly with possible factor-graph representations (illustration from [PP19]).

the upper branch, we get:

$$f_{\text{add}}(x_0, x_1, \hat{x}_0) = \begin{cases} 1 & \text{if } x_0 + x_1\omega = \hat{x}_0 \bmod q \\ 0 & \text{otherwise} \end{cases}$$

As later observed in [PP19], modeling these operations individually causes many small loops in the factor graph which results in reduced BP convergence. Instead, they propose to merge the computations of a butterfly into a single factor node f_{bf}. When combining this optimization with an improved message schedule and a certain amount of message damping, the BP convergence performance can be significantly improved.

3.3 Limitations of Prior Work

While [PPM17, PP19] demonstrate the possibility of side-channel attacks on the NTT, their presented attacks either fall somewhat short of being practical or are only applicable in certain scenarios.

The attack in [PPM17] relies on template matching of modular multiplication operations which requires close to a million different templates. Furthermore, although not strictly required, they also exploit a certain time-invariance of the multiplication operation since it has a data-dependent behavior on their target device. Finally, they analyze a fairly simple and generic NTT implementation that misses several performance optimizations, such as lazy reductions, that are nowadays commonly used.

The attack from [PP19] can be seen as a significant improvement over [PPM17] in terms of practicality, but it also comes with restrictions to certain scenarios. First and foremost, their presented BP improvements allow them to replace value-based templates with generic noisy Hamming weight templates. They also show a practical attack using power traces of an ARM Cortex M4 device. On the downside, they rely on additional information about the narrow support (value range) of polynomial coefficients at the NTT input which is only present during ring-LWE encryption. Even in this case, their evaluation shows that they can only handle noise levels with $\sigma \leq 0.4$ when considering masked implementations.

4 Chosen Ciphertext k-Trace Attack

In this section we present our new attack that overcomes the limitations described in Section 3.3. We first give an outline of the attack, then give details on the different aspects in the subsequent sections.

4.1 Attack Outline - Improving BP for the NTT

Our attack targets Kyber's decryption step, i.e. line 2 of Algorithm 4 with the aim of recovering the victim's long-term private key $\hat{\mathbf{s}}$. Previous work has observed that SASCA can recover the coefficients used in the NTT if sufficient additional information is available [PP19]. We provide that information by way of a Chosen Ciphertext Attack (CCA). Our attack ensures the inverse NTT (NTT^{-1}) will be given sparse input, meaning that most of the NTT coefficients are known to have a value of zero. Our attack works by:

1. Creating a ciphertext $c = (\mathbf{c}_1, c_2)$ such that the inverse NTT operations will be performed on sparse input involving the secret $\hat{\mathbf{s}}$, which combined with the highly structured NTT gives much additional information.
2. Extracting the sparse, secret data from (a) SCA trace(s) on the inverse NTT.
3. Recovering the private key from the recovered information.

In NewHope and Kyber, to decrypt a message the recipient has a secret key $\hat{\mathbf{s}}$ in the NTT domain, and calculates $\mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}})$, where $\hat{\mathbf{u}}$ is the decompressed ciphertext in the NTT domain. Since $\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$ is taken pairwise-pointwise, it will be sparse if $\hat{\mathbf{u}}$ is pairwise-sparse. For NewHope, this is easily achieved because the ciphertext is sent as $\hat{\mathbf{u}}$, i.e. in the NTT domain. For Kyber however, the ciphertext is sent compressed as $\mathbf{c}_1$ in the standard domain (cf. Algorithm 2, l. 5), so it is necessary to find a ciphertext $\mathbf{c}_1$ such that $\hat{\mathbf{u}}$ is sparse and $\mathbf{u} = \mathsf{Decompress}(\mathsf{Compress}(\mathbf{u}))$. In the remainder, we will refer to a ciphertext meeting this condition as *compressible*.

4.2 Creating Sparse NTT Values

We aim to find a compressible ciphertext $c = (\mathbf{c}_1, c_2)$ where $\hat{\mathbf{u}}$ is zero on some subset S of the coefficients. The component $\mathbf{c}_1$ consists of k independent ring elements $c_{1,i} \in \mathcal{R}_{2^d}$. We will explain two methods of forcing the NTT $\hat{u}_i$ of one such component's decompressed image, $u_i \in \mathcal{R}_q$, to be sparse. In the following we will omit the subscript i for clarity.

Note that due to Kyber's field $\mathbb{Z}_q$ lacking a 512th root of unity, its NTT has only 7 layers instead of 8, and the even and odd coefficients never mix. It can therefore be seen as two parallel "half-NTTs" each on 128 coefficients: one half-NTT on the even coefficients and the other half-NTT on the odd coefficients. We can aim for the output of one half-NTT to be sparse, and the other to be zero. Furthermore, the even and odd coefficients will be mixed during the pairwise-pointwise multiplication step, where a pair of one even and one odd coefficient are multiplied by the corresponding position of the NTT'd private key. So we don't need to set half the ciphertext to zero: the attack will work just as well if we set the two halves of the ciphertext so that their half-NTTs have the same sparse support. In any case, working with half-NTTs improves the speed of our methods.

The first method is to solve a short vector problem, since the set of uncompressed ciphertexts u for which $\hat{u}$ are zero on S form a lattice. We have

$$c_1 = \mathsf{Compress}(u) = \left\lfloor u \cdot 2^d/q \right\rceil \mod 2^d,$$

where $d = 10$ for Kyber512 and Kyber768, and $d = 11$ for Kyber1024. We therefore want to construct pairs of vectors $(u, \mathsf{Compress}(u))$ where $\hat{u}$ is sparse and $\tilde{u} = u \cdot 2^d - \mathsf{Compress}(u) \cdot q$ is a short vector. If all the coefficients of $\tilde{u}$ are small then u will be compressible. Using BKZ-2.0 [CN11] with block size 70, we were able to find ciphertexts $(\mathbf{c}_1, c_2)$ where each half-NTT component of $\hat{\mathbf{c}}_1$ is zero in all but 32 out of its 128 positions, and which are compressible with $d = 10$. BKZ is somewhat slow with such a large block size, but it only needs to be run once. For Kyber1024 with $d = 11$, it is possible to generate sparser vectors, with only 16 non-zero coefficients instead of 32. We were able to generate such

vectors in some instances of BKZ-80 by shuffling the rows of the basis randomly. Note however, that an attacker only needs to generate this once in advance, independent of the private key. For better performance of the belief propagation, we further distributed the non-zero coefficients within each vector and matched them pairwise in the second set of 128 coefficients, due to the pointwise-pairwise product.

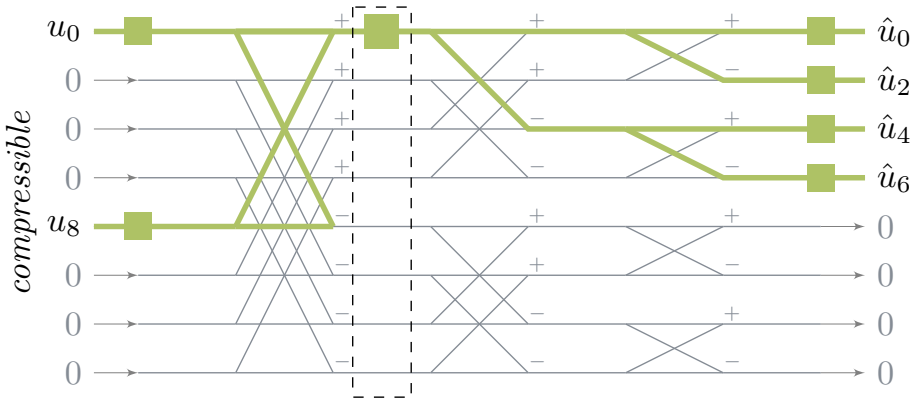


Figure 4.1: Generating sparse NTT values. (simplified for 8 even coefficients and multiplications by ζ omitted) The inputs of the NTT (left) need to be compressible, while the right side should be sparse. A sparse polynomial is found by iterating through a single intermediate value in layer ℓ (here: $\ell = 1$, highlighted by the dashed box) until a compressible input of the NTT is found. This automatically results in a sparse output in the NTT domain (right side). Note, in Kyber the same has to be performed independently for the odd-indexed coefficients.

We also developed a faster approach, which is depicted in Figure 4.1; it takes advantage of the layered structure of the NTT. To achieve a sparse half-NTT output, we set a single intermediate value $\hat{u}_{\ell,j}$ in layer ℓ to a non-zero value, and all other values in that layer to zero. This creates an input with 2^ℓ non-zero coefficients whose half-NTT has $2^{7-\ell}$ non-zero coefficients. Setting $\ell = 2$ for Kyber512 or Kyber768, we find that the resulting 4 coefficients will be compressible with probability of approximately $(2^d/q)^4 \approx 1/112 \approx 30/q$. This estimate indicates that approximately 30 possible intermediates in each position result in compressible ciphertexts. In our experiments we were able to find 32 such possible intermediate values for every position. Each of these results in $2^{7-2} = 32$ out of 128 coefficients in the NTT domain which are non-zero. The positions of these non-zero coefficients are determined by the first ℓ bits of the index $j \in \{0, \ldots, 2^8 - 1\}$. E.g., with $\ell = 2$ and $j = 42 = \texttt{0101010}_2$ the non-zero NTT coefficients are indexed $\texttt{01xxxxxx}_2$. As a result, we can set multiple intermediates within the same block to non-zero values, while still obtaining a sparse half-NTT result. The compressibility is assured, as the coefficients in the normal domain are disjoint. By using this technique, we can produce an exponentially large family of compressible ciphertexts whose NTT is non-zero in 32 of its 128 pairs of coefficients. Most of these ciphertexts have no non-zero coefficients in the normal domain. For Kyber1024, because $d = 11$ instead of 10, we can apply the same technique with $\ell = 3$ to generate compressible ciphertexts with as few as 16 out of 128 coefficients in the NTT domain which are non-zero.

This second approach can generate ciphertexts essentially instantaneously, but the non-zero output coefficients are all in a contiguous block. This leads to worse performance in our belief propagation step, because there are steps where the inputs to a butterfly are statically known to be zero, so the attacker gains no information from that step. However, it also allows faster reconstruction of the key from the partial information gained by the belief propagation step. We attempted to generalize this approach using different decimations of

the NTT. But our attempts didn't work, for reasons described in Appendix C.

Either approach can be applied to each of the k components u_i of $\mathbf{c}_1$, setting ν non-zero coefficients in each of their decompressed NTT-domain representations. In Kyber, the decryption step first take the sum over the components as

$$\hat{w} := \hat{\mathbf{s}}^T \circ \hat{\mathbf{u}} = \hat{s}_0 \circ \hat{u}_0 + \ldots + \hat{s}_{k-1} \circ \hat{u}_{k-1}.$$

If the values $\hat{u}_0, \ldots, \hat{u}_{k-1}$ have non-zero values in the same ν pairs of positions, then $\hat{w}$ will have non-zero values in those positions, which will then contain linear combinations of the coefficients of $\hat{s}_0, \ldots, \hat{s}_{k-1}$. Therefore, if we recover the coefficients of k different $\hat{w}$ values using a side-channel attack on the inverse NTT, we can solve for ν pairs of coefficients of each of $\hat{s}_0, \ldots, \hat{s}_{k-1}$.

Another approach is to set the coefficients of $\hat{u}_0, \ldots, \hat{u}_{k-1}$ to be non-zero in *disjoint* positions. In that case, $\hat{w}$ will have $k \cdot \nu$ pairs of non-zero coefficients, e.g. 96 pairs of non-zero coefficients for Kyber768. With this method the belief propagation will have worse noise immunity, but if successful it will recover ν coefficients from each of $\hat{s}_0, \ldots, \hat{s}_{k-1}$ in a single trace. This will lead to a single-trace attack when the signal-to-noise ratio is high enough.

4.3 Belief Propagation Details

After computing a compressible ciphertext $\mathbf{u}$ that is sparse in NTT domain, we replace the vector $\mathbf{c}_1$ of a honestly generated ciphertext $(\mathbf{c}_1, c_2)$ by the compression of $\mathbf{u}$ and send it to the target device. The leakage in the butterfly operations of the inverse NTT is exploited by template matching, and the resulting probability distributions are used as input to the belief propagation. As $\hat{\mathbf{u}}$ is pairwise-sparse and the multiplication is pairwise-pointwise, the input to the inverse NTT $\hat{w}$ is sparse. This reduces the entropy at variable nodes at positions with $\hat{\mathbf{u}}_{,j} = 0$ to zero. If we are attacking a masked implementation, we perform those steps for both shares and recover the result, $\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$, from the recovered shares.

Apart from attacking the decryption instead of the key generation, our belief propagation is similar to that of Pessl and Primas [PP19], using the merged butterfly nodes but not applying message damping. Also, in the case of masking, we did not improve our belief propagation by adjoining the graphs for both shares. As the coefficients of the input to the inverse NTT, $\hat{w} = \hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$ are not small and the coefficients of the output of the inverse NTT, $w = \mathbf{s}^T \circ \mathbf{u}$, are given by $\mathbf{s}$ convoluted with $\mathbf{u}$, factor nodes ensuring the consistency for each coefficient individually seem infeasible. Therefore, our belief propagation graphs for both shares are completely independent. As Kyber's NTT uses only 7 instead of 8 layers, each share consists of two separate connected graphs, resulting in four separate graphs for the masked case and two separate graphs for the unmasked graph.

4.4 Recovering the Private Key from Secret Data

The last step of our algorithm is to recover the private key $\mathbf{s}$ from the recovered secret data. This recovered data consists of the value $\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$, where $\hat{\mathbf{u}}$ is sparse, having only n non-zero pairs of values contained in some set S. Since all non-zero elements of $\mathbb{F}_{q^2}$ are invertible, we can divide by $\hat{\mathbf{u}}$ to obtain the corresponding pairs of coefficients in $\hat{\mathbf{s}}$.

We therefore know ν coefficients of $\hat{\mathbf{s}}$, and we want to recover $\mathbf{s}$, which we know has small coefficients. Again, the even and odd coefficients don't interact, so we can consider a 128-element half-NTT. In Kyber768 and Kyber1024 the coefficients of $\mathbf{s}$ are in $\{0, \pm 1, \pm 2\}$, so there are 5^{128} possibilities, and in Kyber512 they are in $\{0, \pm 1, \pm 2, \pm 3\}$, leading to 7^{128} possibilities. The solution is likely to be nearly unique if $q^n > 5^{128}$ or $q^n > 7^{128}$, meaning that at least 26 or 31 coefficients are recovered, respectively. As with finding sparse ciphertexts, this can be written as a shortest vector problem and solved with BKZ.

As before, there is also a more efficient approach when S is a contiguous block. If the coefficients of $\hat{s}$ in a $2^{7-\ell}$-sized block are all known, then we can unwind the last $7-\ell$ steps of the half-NTT, and learn the intermediate values after the first ℓ steps, as depicted in Figure 4.2. Note that this is possible as blocks only depend on known coefficients for the last $7-\ell$ layers. Each of these known intermediate values is a function of 2^{ℓ} unknown coefficients of s, independent of the other intermediates.

The coefficients of s are sampled from a small distribution, so they can be recovered by an exhaustive attack, or more simply a look-up table. We can use $\ell = 2$, where each intermediate depends on $2^2 = 4$ coefficients of s, and each coefficient is drawn from a set of size 5 for Kyber768 or Kyber1024, or 7 for Kyber512. So we can recover each coefficient of s using a lookup table or search of size 5^4 or 7^4 for these two cases.

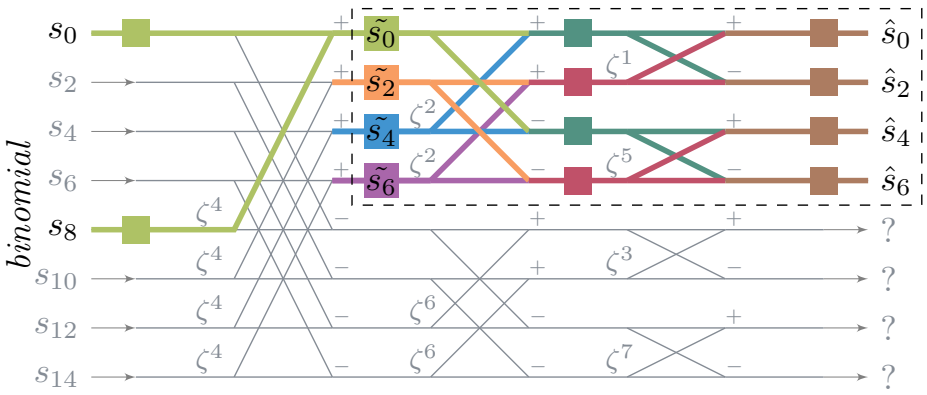


Figure 4.2: Recovering the private key from partial knowledge of $\hat{s}$. After inference of e.g. the upper half of $\hat{s}$, the original key can be recovered as follows. First note, that the last layers of the NTT can be reversed inside the upper half up to the intermediate polynomial $\tilde{s}$. From here, each coefficient can be independently brute forced by its original inputs, which are sampled from small binomial distributions, e.g. $\{-2, \ldots, 2\}$. Here $\tilde{s_0} = s_0 + \zeta^4 s_8$ which results in only $5^2 = 25$ possible combinations.

For Kyber1024, it is possible to generate sparser vectors, with only 16 pairs of non-zero coefficients per polynomial instead of 32 (see Section 4.2). This allows an attack with an even lower signal-to-noise ratio. However, knowledge of only 16 pairs of coefficients of $\hat{s}$ does not yield a unique solution; instead there are more than 100 possibilities of the 8 coefficients of s for each of the 16 intermediates. Even by sorting by likelihood of the SCA results, we could not reduce this to a computational feasible level. Therefore, for this variant of the attack, we need 8 traces instead of 4, so that we recover 32 pairs of coefficients of $\hat{\mathbf{s}}$ per polynomial.

5 Results

First, in Section 5.1 results showing the effectiveness and allotted noise levels for our attack variants are presented. Additionally, we provide estimates for the remaining security after partial-key recovery in Section 5.2.

5.1 Key Recovery Attack

We evaluate the full attack strategy on Kyber512, Kyber768 and Kyber1024 via simulated leakage experiments. The choice of relying on simulated experiments is mainly motivated by (1) easy reproducibility and comparability of our results, (2) the fact that practical

attacks have already shown in a similar attack setting. More precisely, the authors of [PP19] have shown that SPA attacks on the NTT operation are possible on a 32-bit STM32F405 microprocessor if the σ in a corresponding noisy Hamming weight leakage simulation is below 2 (unmasked scenario). Our attacks supersede these results both in terms of noise resistance and versatility. As we will show, our attacks can work up to noise level of about $\sigma = 3.1/2.7$ in unmasked/masked scenarios and are, in contrast to [PP19], applicable in public-key decryption scenarios. The codebase itself is written in Rust and Python.

Leakage Model In the noisy Hamming weight leakage model an attacker can observe the Hamming weight (HW) with an additive Gaussian noise of certain intermediate variables of an analyzed computation. More precisely, for an intermediate a the simulated leakage results in $\mathrm{HW}(a) + \mathcal{N}(0, \sigma)$, with $\mathcal{N}$ being a normal distribution with a mean of zero and standard deviation σ. In a recent paper [KPP20] the authors performed actual power measurements to show that this leakage model is a quite suitable approximation for load/store instructions on current microcontrollers. For an 8-bit target (XMEGA 128D4) their measured leakage closely matches the noisy Hamming weight model with a σ of 0.5. In the 32-bit scenario (STM32F405) the authors measured a σ in a broader range between 0.4 in the best case and 3.0 in the worst case. These numbers can be lowered by averaging multiple measurement traces (10 traces have been averaged in their evaluation) to a σ in the range of 0.2 to 1.3. Please note that averaging is only possible in an unmasked setting.

Target Implementation Details Our simulation is based on the current Kyber reference implementation [ABD+], which is very similar to the implementation targeted in [PP19], allowing for direct comparison. We re-implemented the relevant functions in python/numpy to generate simulated noisy HW leakages of the relevant 16-bit signed integer values. In practice this step would require building templates of the load/store instruction for the HWs from 0 to 16. The leakage is taken of the intermediate values between each layer, i.e. the input to each butterfly operation (cf. Figure 3.1c). Analog to [PP19] we target the load (LDR) and store (STR) operations of the individual coefficients, which corresponds to the above-mentioned leakage model. Note that in contrast to [PP19] we target the inverse NTT in the Kyber decryption (cf. Algorithm 4, l. 2).

To the best of our knowledge, there is no published masked implementation specifically for Kyber. Therefore, we consider a masked implementation that follows the generic ring-LWE masking strategy from [RRdC+16, OSPG18], which is also summarized in Section 2.4. Hence, the secret key $\widehat{\mathbf{s}}$ is assumed to be additively masked in two shares, $\widehat{\mathbf{s}} - \widehat{\mathbf{m}}$ and $\widehat{\mathbf{m}}$, with the coefficients of $\widehat{\mathbf{m}}$ sampled uniformly from $\{-(q-1)/2, \ldots, (q-1)/2\}$. A simplified depiction of masked Kyber decryption is shown in Figure 5.1.

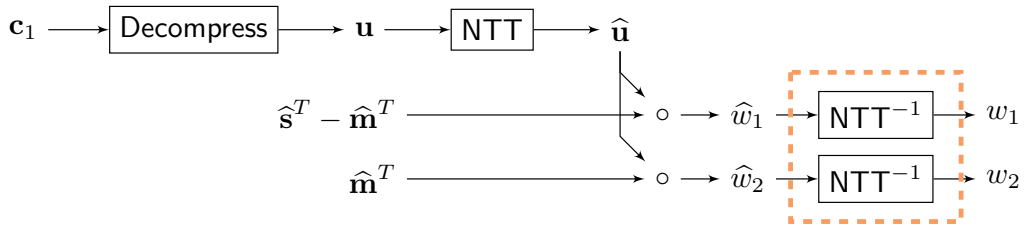


Figure 5.1: Simplified depiction of a masked Kyber decryption operation. Parts that are unnecessary for our analysis are omitted. The side channel leakage is taken from the NTT^{-1}, highlighted by the dashed orange box.

Belief Propagation Instantiation We built an optimized, multi-threaded implementation of belief propagation in Rust. We structured the graph according to our Python simulation

of the inverse NTT, with the masking split in two independent shares. We highlight further algorithmic optimizations in Section 4.3. As discussed in Section 3.1, loopy belief propagation needs break conditions. We employ a number of conditions based on empirical experiments to allow for a reasonable trade-off between inference and runtime. We set the maximum number of iterations to 1000. Additionally, we abort if the Shannon entropy of all nodes is less than 0.1 bits, or the entropy change is less than 0.05 bits after 20 iterations. We further abort if after 200 iterations less than one more correct coefficient became the most probable or we found all correct coefficients, which requires knowledge of the secret and hence would not be available to an attacker. With these break conditions, a belief propagation run takes on average 20 minutes using two Intel Xeon E5-2650 v4 2.20GHz with 24 cores and hyper-threading.

Attack Results The chosen ciphertext $\mathbf{c_1}$ is generated according to Section 4.2, resulting in a sparse vector $\hat{\mathbf{u}}$ generated according to the two approaches. First, the sparse vectors generated with BKZ allowed us to distribute the non-zero coefficients. Note however, due to the nature of the NTT in Kyber, the coefficients are only distributable pairwise. Second, we used the faster generation approach depicted in Figure 4.1, which results in the non-zero coefficients aligned in contiguous blocks. The number of non-zero coefficients per polynomial $\hat{u}_i$ could be set to either 256, 128, 64, and 32, with the last only applicable for Kyber1024. Combining these polynomials in the vector $\hat{\mathbf{u}}$ results in sparse inputs to the inverse NTT $\hat{w} = \hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$ with 256, 192, 128, 64, and 32 non-zero coefficients. Note that for the final key recovery it is important in which vector components the non-zero coefficients are placed in order to reduce the required number of attack traces (see *Final Key Recovery* below). Also, further intermediate combinations with 32 non-zero coefficients (e.g. 96 non-zero coefficients) were omitted, as it is only applicable to Kyber1024 and would only marginally reduce the number of traces needed (e.g. 3 instead of 4) for the final key recovery.

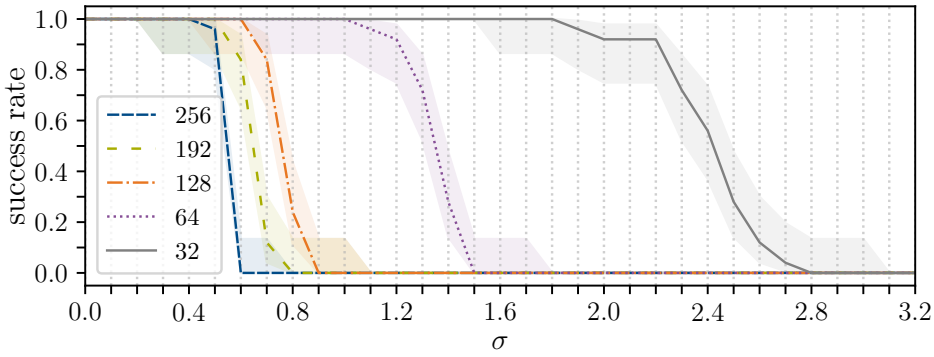
We ran experiments for a range of σ from 0 up to 3.2 in steps of 0.1. For every value of σ in the relevant range where the observed probability of success was not 0 or 1, we repeated the experiments 25 times. We used the same strategy in both the masked and unmasked scenario and the two different sparse vector sets. After reaching our abort condition, we compared the output of the belief propagation to the correct secret key, and counted the individual run as successful if all correct coefficients had the highest probability.

The results for the distributed non-zero coefficients are shown in Figure 5.2 with the legend highlighting the number of non-zero coefficients. The shaded area around each of the lines represents the confidence interval of the experiments for a confidence level of 95%. Since the number of experiments is not large, we use the Wilson score interval [Wil27] that performs better than the normal approximation interval in such cases.

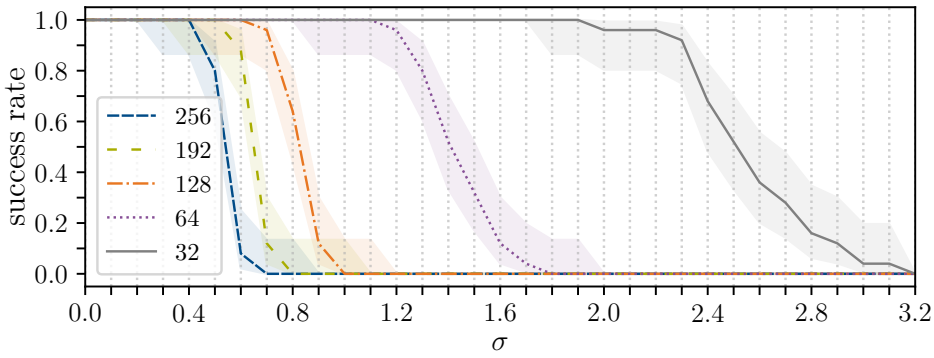
We draw the following conclusions from the experiments. In the non-sparse case (256 non-zero coefficients) we observed a success rate of 1 only for $\sigma \leq 0.4$. This agrees with the Hamming weight leakage model results of [PPM17], targeting a 256-coefficient ring-LWE inverse NTT. Applying our sparseness strategy to the input ciphertext c, we can increase the noise tolerance significantly. For example, with 64 distributed non-zero NTT coefficients, the noise level can go up to $\sigma = 1.2$, while the probability of success stays within a confidence interval of 0.75 to 0.97 for the masked and 0.80 to 0.99 for the unmasked version.

By setting even more coefficients to zero, the noise level can be further increased, while keeping a solid success rate. For example, for Kyber1024 by setting all but an eighth (i.e. 32) of the coefficients to zero with BKZ, at $\sigma = 2.2$ the confidence interval is 0.75 to 0.98 for the masked and 0.80 to 0.99 for the unmasked version.

As the graphs show, we can further increase the achievable σ by allowing a lower



(a) With masking



(b) Without masking

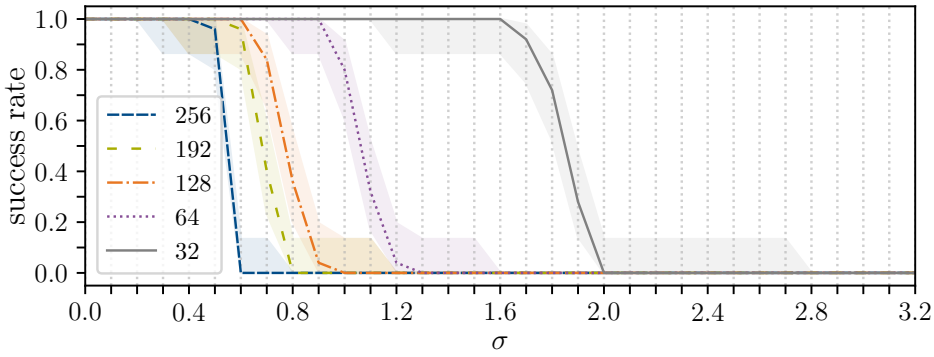
Figure 5.2: Attack results for different noise levels σ with distributed non-zero coefficients. The figures show the attack success rate for the masked (a) and unmasked (b) implementations, where the sparse vectors are generated with BKZ. Each data point within the relevant area is the average of 25 runs with a step-size of 0.1 for σ. The shaded area marks the 95% confidence interval. It can be seen that with a decreasing number of non-zero coefficients (given in the legend) the achievable noise tolerance is significantly increased.

success rate. Note however, that as the success rate reduces, the required number of attack traces will increase as we have to evaluate several runs until we find one where we achieve successful convergence for the belief propagation.

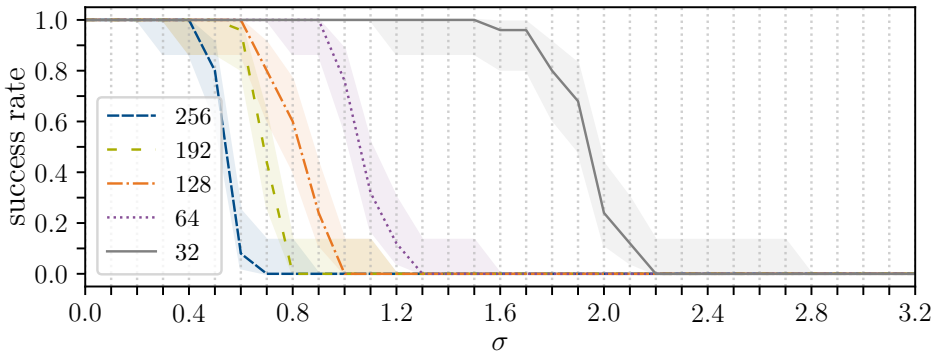
Our attack shows similar results for the masked and unmasked case, because each masking share can be attacked individually with the same technique as in the unmasked case. In particular, as can be seen from Figure 5.1, masking the secret key as $(\widehat{\mathbf{m}}, \widehat{\mathbf{s}} - \widehat{\mathbf{m}})$ does not influence the sparsity of $\widehat{\mathbf{u}}$. Further, if $\widehat{\mathbf{u}}$ is sparse then $\widehat{\mathbf{m}} \circ \widehat{\mathbf{u}}$ and $(\widehat{\mathbf{s}} - \widehat{\mathbf{m}}) \circ \widehat{\mathbf{u}}$ are both sparse with the same support. We can separately recover them with belief propagation, and we add them together to obtain the (again sparse with the same support) $\widehat{\mathbf{s}} \circ \widehat{\mathbf{u}}$. From this step onwards the attack continues as in the unmasked case. We then run our sparse key recovery on that value.

Also note that the mask is thus removed in each attack trace, allowing us to repeat and combine the coefficients recovered in multiple traces as in the unmasked case.

The results for sparse vectors with non-zero coefficients in contiguous blocks (see Figure 4.1) are shown Figure 5.3. Here, with 64 non-zero coefficients in a contiguous block, our approach still shows a non-zero success rate up to $\sigma = 1.2$, with a success rate confidence above 0.87 up to $\sigma = 0.9$. This can again be increased for Kyber1024 up



(a) With masking



(b) Without masking

Figure 5.3: Attack results for different noise levels σ with non-zero coefficients in contiguous blocks. The figures show the attack success rate for the masked (a) and unmasked (b) implementations with the non-zero values in a contiguous block (see Figure 4.1). Each data point within the relevant area is the average of 25 runs with a step-size of 0.1 for σ. The shaded area marks the 95% confidence interval. In comparison to Figure 5.2, the noise tolerance is mainly decreased for a low number of non-zero coefficients (i.e. 64 and 32). Note, that the graph for 256 non-zero coefficients is identical to Figure 5.2, as this is the non-sparse case.

to a $\sigma = 1.7$ by reducing the number of non-zero coefficients to 32, with a success rate confidence between 0.80 and 0.99 for unmasked, and between 0.75 and 0.98 for masked. For larger numbers of the non-zero coefficients the faster generation of the sparse vectors did not decrease the noise tolerance significantly. Note, that for 256 non-zero coefficients the graphs are identical as for the distributed sparse coefficients, as the location of non-zero coefficients is not relevant in this non-sparse case.

In [PP19] the unmasked case allowed a success rate above 0.9 up to $\sigma = 1.5$, slightly exceeding our achieved noise tolerance for Kyber512 and Kyber768. This is mainly because [PP19] targets the NTT in the encryption, whereas we target inverse NTT in decryption. In the former case, the input distribution is sampled from a small binomial distribution, which acts as an additional constraint for belief propagation (cf. Section 3.3). But our attack extract the long-term secret key $\mathbf{s}$, in comparison to the ephemeral secret $\mathbf{r}$. Additionally, the advantage of [PP19] disappears in the masked setting, as here the inputs the NTT are not small, dropping the success threshold to $\sigma \leq 0.4$.

Table 5.1: Overview of the required number of traces for a full secret key recovery given the number of non-zero coefficients and the resulting noise tolerance level for a Success Rate (S.R.) > 0.7 and > 0 for all Kyber security levels with masking. The first number is with sparse vectors generated with BKZ, the second with easier generation with the non-zero coefficients in contiguous blocks. As the final key recovery for Kyber512 using BKZ is computationally expensive, an attacker could opt for an extra attack trace (numbers in brackets) for a fast final key recovery. The k-trace attack ($k \in 2, 3, 4$) is applicable with 64 non-zero coefficients and a noise level of 1.2|0.9 or 1.4|1.2.

Sparseness	**Kyber512**	**Kyber768**	**Kyber1024**	**S.R. > 0.7**	**S.R. > 0**
# non-zero coeffs.	# traces	# traces	# traces	max. σ	max. σ
32	–	–	8	2.2 | 1.7	2.7 | 1.9
64 − k-trace attack −	**2** (3)	**3**	**4**	1.2 | 0.9	1.4 | 1.2
128	1 (2)	2	2	0.6 | 0.6	0.8 | 0.9
192	1	1	2	0.5 | 0.6	0.7 | 0.7
256	1	1	1	0.5 | 0.5	0.5 | 0.5

Final Key Recovery As a final attack step, we implemented key recovery according to Section 4.4, inverting the pairwise-pointwise scalar product and the half-NTT. For a simpler separation of the k dimensions, we opted for non-zero values in disjoint positions for each dimension. This allows us to directly recover the corresponding segment of coefficients of $\hat{\mathbf{s}}$ by dividing out the non-zero section of our sparse vector $\hat{\mathbf{u}}$.

For the case of pairwise distributed non-zero coefficients in the sparse polynomials u_i, this can be written as a shortest vector problem and solved with BKZ. With a block size 70 we were able to solve 92.7% out of 590 attempts for a half-vector component of Kyber768/Kyber1024 with 32 out of 128 non-zero coefficients. By increasing the block size to 80, we were able to increase the recovery success rate to 100% out of 100 attempts. For Kyber512 the task is more difficult due to the larger binomial distribution, but we are still able to succeed in 54% out of 100 attempts with block size 80. This can most likely be increased further by larger block sizes, however an alternative would be to increase the number of coefficients recovered to 48 out of 128. With this we could solve it in 100% out of 100 attempts, employing only BKZ-40. Hence, for Kyber768 and Kyber1024 an attack yielding a total of 64 coefficients per vector component allows for a full key recovery. For Kyber512 the computational cost for key recovery was only feasible for us in half of the cases, but by increasing the number of coefficients recovered to 96 per vector component, only minimal efforts are needed for a full key recovery.

In the case of the non-zero coefficients in a contiguous block, the faster approach highlighted in Figure 4.2 is possible. We ran 1000 experiments for each Kyber parameter set, in which we attempted to determine the key from a recovered $\hat{w} = \hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}$, with $k \cdot 64$ non-zero coefficients in $\hat{\mathbf{u}}$. For Kyber768 and Kyber1024 the key was uniquely determined in all our tests. For Kyber512, a few coefficients of each key were not uniquely determined. This resulted in 2^{12} possible values for the entire private key on average, up to a maximum of 2^{22} in our experiments. However, the private key could be recovered by checking these possibilities against the known public key. In total, key recovery from $k \cdot 64$ known coefficients of $\hat{\mathbf{s}}$ took at most a few seconds on a laptop.

Note that for each dimension, 64 coefficients suffice to uniquely recover all coefficients of $\mathbf{s}$. Here, a trade-off has to be taken between the sparseness in the inverse NTT and the number of coefficients recovered with each trace. In Table 5.1 the necessary number of traces needed is summarized for the different security levels of Kyber together with our achieved noise tolerance threshold on σ for a masked implementation. The first numbers are for the sparse vectors generated with BKZ, with the non-zero coefficients distributed

pairwise to improve the belief propagation. The second numbers show the results with sparse vectors generated with the faster approach employing the butterfly structure of the NTT, here the non-zero coefficients are in contiguous blocks. With the contiguous block sparseness we are able to perform a successful single-trace attack on Kyber512 up to a σ of 0.6 and fully recover the secret key in a $2k$-trace attack for Kyber1024 up to a σ of 1.7 (Success Rate S.R. above > 0.7). The k-trace attack with full key recovery is possible up to a σ of 0.9 for all Kyber security levels. By generating the sparse vectors with BKZ this can be increased up to a σ of 1.2. Note however, for Kyber512 using BKZ for the final key recovery step, the number of traces might need to be increased by one (numbers in brackets), in order to allow for faster solving for the original key. With repeating failed runs (Success Rate S.R. between $0 <$ S.R. < 0.7) we can further increase the noise level in the k-trace attack setting up to $\sigma \leq 1.4$ and for Kyber1024 even to $\sigma \leq 2.6$ in the $2k$-trace attack setting. All results are given considering masking of the secret key $\mathbf{s}$.

5.2 Security Estimates from Partial Key-Recovery

For full key recovery, the attack presented in Section 4 either requires a single chosen ciphertext trace to be measured with low noise ($\sigma \leq 0.5 - 0.7$) or k traces with a noise up to $\sigma \leq 1.2$. We saw the effectiveness of such an attack in the previous section. However, it might not always be possible or practical to get the full k traces in the case of an assumed noise level of $\sigma \approx 1.2$. Therefore, an estimate of the remaining security after less than k traces is shown in the following, assuming 64 coefficients recovered per trace.

To estimate the remaining security in these cases we use the same methodology as was used for the security estimates of Kyber; core-SVP hardness for the primal attack. Note that core-SVP estimates are particularly conservative, and a b-bit security estimate is not equivalent to a b-bit remaining key entropy. As is also noted in [ABD+17], more refined estimates of the security can be made. Since this is not the purpose of this paper, we restrict ourselves to the estimates below.

In Table 5.2 it can be seen that even with a single trace recovered in the k-trace attack, the estimate for the remaining security shows a significant drop. We emphasize again that these numbers do not necessarily mean that 2 traces in the k-attack on Kyber768 leads to a practical break of the full Kyber key. Even for the 2 traces, sieving records are currently still well out of range recovering a short vector of a 1023-dimensional lattice [DSvW21].

Table 5.2: Security estimates for the remaining security after partial k-trace key recovery for Kyber768 ($k = 3$), assuming 64 non-zero coefficients of $\hat{\mathbf{s}}$ recovered per trace.

# **Traces**	0	1	2	3
Dimension d	1423	1238	1023	0
BKZ-blocksize β	625	381	152	–
core-SVP (classical)	182	111	44	0
core-SVP (quantum)	165	101	40	0

Recovery of LWE keys from partial key recovery is out-of-scope of this paper, but is an active area of research. E.g. in [DDGR20] it was shown that knowledge of a partial key can be incorporated into the u-SVP lattice of the LWE key recovery. We leave the adaptation of these techniques to the module-LWE setting as future work.

We do however see that the security estimates for classical and quantum hardness for 1 trace already drop well under the desired 128-bit security levels, and for 2 traces gives an alarming drop in security. For Kyber512 similar estimates shown that even 1 trace reduces the security estimates from 118 / 107 bits of security to 47 / 43 bits in the respectively classical / quantum setting. For Kyber1024, 1 trace in the k-trace attack

drops the security to the Kyber768 case. From thereon the reduction of security declines comparably to Kyber768. For completeness the full table is included in Appendix B.1.

6 Discussion

In the previous sections, we demonstrated how simple power analysis attacks on lattice-based cryptographic schemes can be significantly improved over previous works, both in terms of noise resistance and wider applicability. We now briefly discuss how our attack could be applied to lattice-based schemes other than Kyber and NewHope, and options for countermeasures against our attack.

Application to other Schemes Kyber and NewHope explicitly mention the usage of NTT computations in their specification, and therefore Kyber was an obvious focus of our presented attack. However, it has been shown in [CHK+21] that the NTT can be used to implement the polynomial multiplication of NIST finalists Saber [DKRV18] and NTRU [CDH+20]. The same is the case for NIST alternate NTRU Prime [BCLvV17] in [ACC+21b]. Even though these schemes operate in rings that are less "NTT-friendly", their polynomial multiplication can be lifted to a larger ring (in degree and/or modulus) that both allows the application of an NTT and ensures that reductions do not affect the correctness of the result.

We also conjecture that belief propagation techniques presented in this paper could increase the effectiveness of simple power analysis attacks on other systems. Different multiplication algorithms that process secret coefficients in blocks (e.g. Karatsuba [KO63] or Toom-Cook [Too63, Coo66]) can likely be forced to have sub-blocks be "special", e.g. small or sparse. In this case, belief propagation can be used to learn more information from side channels, similarly to the attack presented in this paper. The structures of underlying rings like existing automorphism and sub-rings might also help the effectiveness of belief propagation, however this is left for future work.

Application to other Implementations The choice of our target implementation (and leakage model) is mainly motivated by the fact that we want to allow accurate comparisons with previous works [PPM17, PP19, KPP20], and by the fact that the attack in [PP19] has already been reproduced on a real Cortex-M4 device. Nevertheless, it is natural to ask how our attack could be adapted to more recent, optimized Kyber implementations such as the one from `pqm4` [KRSS]. This implementation mainly differs from the implementation that is considered in this paper in two aspects. First, the `pqm4` implementation stores two NTT coefficients within one 32-bit word and then uses vectorized instructions such as `uadd16` to perform two halfword additions concurrently. Second, the `pqm4` implementation uses a register allocation strategy that reduces the amount of load/store instructions to only occur every third NTT layer.

When adapting the factor graph to the situation where two NTT coefficients are stored within the same word, one could make use of a strategy that is already used in the single-trace attack on 32-bit implementations on Keccak in [KPP20]. There, the authors use a clustering approach to represent one 32-bit word as two halfwords in the factor graph, not because the algorithmic description of Keccak requires it, but because BP runs into serious runtime issues when performing message passing for 32-bit variable nodes. This will however come at the cost of reduced convergence performance.

To accommodate for a potential lack of load/store instructions, one could instead opt for templating the multiplication with twiddle factors, as done in [PPM17]. This will likely increase the complexity of template generation to some extent but would also have the advantage that multiplications need to be executed separately for each halfword,

eliminating the need for clustering. We leave a more concrete evaluation of our attack approach against more optimized implementations for future work.

Masking of the Input Masking is first and foremost a countermeasure against differential attacks but also somewhat affects the performance of profiled attacks. Most notably, masking randomizes data during computations, which effectively prevents an attacker from performing averaging. Consequently, side-channel attacks in this setting either need to work with single measurements or find ways to combine information from multiple computations using different masks. In our case, we are able to retrieve the unmasked intermediate values by individually attacking each masking share. However, this is possible for each trace individually, and we are thus able to combine coefficients recovered from multiple traces also in the masked case. Hence, our attack shows similar results for the masked and unmasked case (cf. Figure 5.2). It is important to point out that our attack on Kyber is only applicable for masking schemes that mask the key but not the input (during decryption), which is the common case [RRdC+16, OSPG18]. With masking of the input, however, our assumption of a sparse input to the inverse NTT does not hold anymore. In this case, our noise tolerance is reduced to the non-sparse case (cf. Figure 5.2 with 256 non-zero coefficients).

Hiding As mentioned in previous works that study belief propagation based side-channel attacks, a rather straight forward and effective protection against such attacks can be achieved by using hiding techniques such as shuffling [PPM17, PP19, KPP20, RPBC20]. By randomizing the order of executed operations within an NTT computation, leakage points cannot be trivially assigned to the correct variable nodes anymore. This then leads to contradictions during belief propagation and prevents convergence. In a similar spirit, the insertion of random dummy operations inside the NTT can increase the difficulty of attacks.

7 Conclusion

We presented a method for crafting ring/module-LWE ciphertexts that result in sparse polynomials at the input of inverse NTT computations, and present a novel attack that uses this sparseness to significantly improve side-channel attacks. Our attack shows that side-channel security of lattice-based schemes cannot be neglected and that relying on (key) masking alone does not offer protection at a reasonable cost. While this work focuses on Kyber, variations of our attack are also applicable to other lattice-based schemes like NewHope, and potentially to implementations of NTRU, SABER, LAC etc. which use Number Theoretic Transforms.

Acknowledgements

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported by the German Ministry of Education, Research and Technology in the context of the project Aquorypt (reference number 16KIS1018 and 16KIS1017K) , and by the Austrian Research Promotion Agency (FFG) via the K-project DeSSnet, which is funded in the context of COMET – Competence Centers for Excellent Technologies by BMVIT, BMWFW, Styria and Carinthia, as well as by the European Union’s Horizon 2020 research and innovation programme under grant agreement No 830927. Part of this work was done during the Lorentz Center Workshop "Post-Quantum Cryptography for Embedded Systems", Oct. 2020.

References

[ABD+] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER reference implementation. Available at https://github.com/pq-crystals/kyber/, commit '8e9308bd0f25fa698e4f37aba216249261f8b352', last accessed 2021-04-11.

[ABD+17] Roberto Avanzi, Joppe Bos, Láo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, , and Damien Stehlé. CRYSTALS–kyber: Algorithm specification and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2017. https://cryptojedi.org/papers/#kybernist.

[ACC+21a] Erdem Alkim, Dean Yun-Li Cheng, Chi-Ming Marvin Chung, Hülya Evkan, Leo Wei-Lun Huang, Vincent Hwang, Ching-Lin Trista Li, Ruben Niederhagen, Cheng-Jhih Shih, Julian Wälde, and Bo-Yin Yang. Polynomial multiplication in NTRU prime comparison of optimization strategies on Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):217–238, 2021.

[ACC+21b] Erdem Alkim, Dean Yun-Li Cheng, Chi-Ming Marvin Chung, Hülya Evkan, Leo Wei-Lun Huang, Vincent Hwang, Ching-Lin Trista Li, Ruben Niederhagen, Cheng-Jhih Shih, Julian Wälde, and Bo-Yin Yang. Polynomial multiplication in NTRU prime. 2021(1):217–238, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8733.

[BBC+20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[BBPS19] Madalina Bolboceanu, Zvika Brakerski, Renen Perlman, and Devika Sharma. Order-lwe and the hardness of ring-lwe with entropic secrets. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 91–120. Springer, 2019.

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, 2017.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

[BPO+20] Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):483–507, 2020.

[CDH+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[CHK+21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for NTT-unfriendly rings new speed records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):159–188, 2021.

[CN11] Yuanmi Chen and Phong Q Nguyen. BKZ 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2011.

[Coo66] Stephen A. Cook. *On the minimum computation time of functions*. PhD thesis, 1966. URL: `http://cr.yp.to/bib/entries.html#1966/cook`.

[CT65] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[DGKS20] Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. (in)security of ring-lwe under partial key exposure. *J. Math. Cryptol.*, 15(1):72–86, 2020.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2018*, pages 282–305, Cham, 2018. Springer International Publishing.

[DSvW21] Léo Ducas, Marc Stevens, and Wessel van Woerden. Advanced lattice sieving on gpus, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 249–279, Cham, 2021. Springer International Publishing.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptol.*, 26(1):80–101, 2013.

[GRO18] Joey Green, Arnab Roy, and Elisabeth Oswald. A systematic study of the impact of graphical models on inference-based attacks on AES. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, volume 11389 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2018.

[GS66] W. M. Gentleman and G. Sande. Fast fourier transforms: For fun and profit. In *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS '66 (Fall), page 563–578, New York, NY, USA, 1966. Association for Computing Machinery.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KO63] Anatoly A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963. URL: `http://cr.yp.to/bib/entries.html#1963/karatsuba`.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[Mac03] David J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[Nat] National Institute of Standards and Technology. Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure Saber KEM. *IACR Cryptol. ePrint Arch.*, 2021:79, 2021.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.

[PP19] 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, Santiago de Chile,*

Chile, October 2-4, 2019, Proceedings, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[RBRC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) NIST pqc candidates for practical message recovery and key recovery attacks. Cryptology ePrint Archive, Report 2020/1559, 2020. https://eprint.iacr.org/2020/1559.

[RdCR+16] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-lwe masking. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2016.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable sca countermeasures against single trace attacks for the ntt. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 123–146, Cham, 2020. Springer International Publishing.

[RRdC+16] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-lwe. *J. Cryptogr. Eng.*, 6(2):139–153, 2016.

[RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-lwe implementation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 683–702. Springer, 2015.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[Too63] Andrei L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.

[VGS14] 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 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[Wil27] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[ZCH+20] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, Oussama Danba, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

A Belief Propagation

What follows is a more thorough description of the belief propagation (BP) algorithm that was taken from [PP19]:

BP allows efficient marginalization in certain probabilistic models. Given is a function $P^*(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m)$, which is defined over a set of N variables $\mathbf{x} \equiv \{x_n\}_{n=1}^{N}$ and the product of M factors. Each of the factors $f_m(\mathbf{x}_m)$ is a function of a subset $\mathbf{x}_m$ of $\mathbf{x}$. The problem of marginalization is then defined as computing the marginal function $Z_n(x_n) = \sum_{\{x_{n'}\}, n' \neq n} P^*(\mathbf{x})$, or the normalized version $P_n(x_n) = Z_n(x_n)/Z$, with $Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x})$.

BP solves this task efficiently by exploiting the known factorization of P^*. First, it represents the factorization in a probabilistic graphical model called factor graph. Factor graphs are comprised of variable nodes, each representing one variable $x_n \in \mathbf{x}$, and factor nodes, each representing one f_m. Factor f_m and variable x_n are connected in the graph if f_m depends on x_n. Second, it performs message-passing on the factor graph. Concretely, it iteratively runs the following two steps until convergence is reached:

1) from variable to factor:

$$u_{n \to m}(x_n) \;=\; \prod_{m' \in \mathcal{M}(n) \setminus \{m\}} v_{m' \to n}(x_n), \tag{3}$$

where $\mathcal{M}(n)$ denotes the set of factors in which n participates.

2) from factor to variable:

$$v_{m \to n}(x_n) \;=\; \sum_{x_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m) \setminus m} u_{n' \to m}(x'_n) \right), \tag{4}$$

where $\mathcal{N}(m)$ denotes the indices of the variables that the m-th factor depends on and $\mathbf{x}_{m \setminus n}$ denotes the set of variables in $\mathbf{x}_m$ without x_n.

After convergence, the marginal function $Z_n(x_n)$ can be computed by multiplying all incoming messages at each node: $Z_n(x_n) = \prod_{m \in \mathcal{M}(n)} v_{m \to n}(x_n)$. The normalized marginals are given by $P_n(x_n) = Z_n(x_n)/Z$, where $Z = \sum_{x_n} Z_n(x_n)$.

B Remaining Entropy Partial Key-Recovery for Kyber1024

Table B.1: Security estimates for the remaining security after partial k-trace key recovery for Kyber1024 ($k = 4$), assuming 64 non-zero coefficients of $\hat{\mathbf{s}}$ recovered per trace.

# Traces	0	1	2	3	4
Dimension d	1885	1680	1495	1280	0
BKZ-blocksize β	877	625	381	152	–
core-SVP (classical)	256	182	111	44	0
core-SVP (quantum)	232	165	101	40	0

C Failed approach: custom decimations

We had hoped to use a different decimation of the NTT as shown in Figure C.1, to produce non-zero coefficients in a pattern other than a contiguous block. For example, Kyber is implemented using decimation-in-frequency, so we might try to implement our technique on a decimation-in-time implementation of NTT. It is also possible to use a custom decimation, where some steps of the NTT are decimated in time and others in frequency.

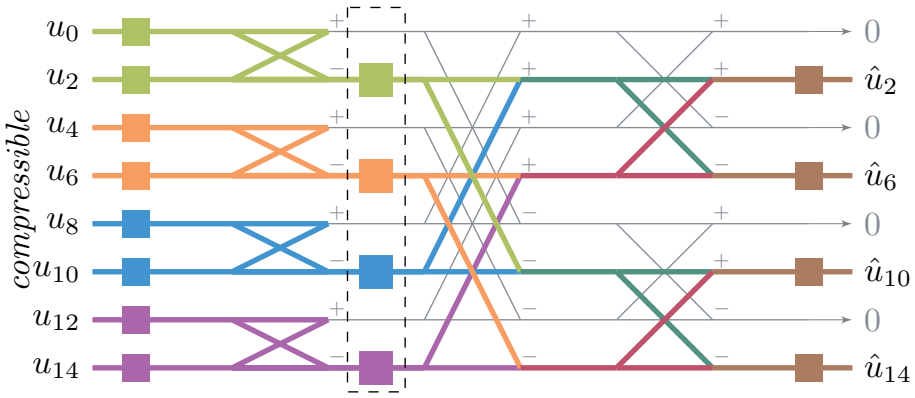


Figure C.1: Rearranged generation of sparse inputs. By rearranging the layers of the NTT, i.e. creating a custom decimation, we hoped to create a different distribution of non-zero values in the sparse NTT. This doesn't work, because performing the layers in the order shown doesn't implement an NTT.

However, this approach doesn't work, because decimation-in-time and decimation-in-frequency are actually the same algorithm, differing only in memory layout, and the same is true for a custom decimation. Changing the computation graph, so that different intermediates are produced, doesn't compute the NTT. Perhaps there is another way to make this technique work, but we didn't find one.