More Practical Single-Trace Attacks on the Number Theoretic Transform

Peter Pessl, Robert Primas

Graz University of Technology, Austria
peter.pessl@iaik.tugraz.at,rprimas@gmail.com

Abstract. Single-trace side-channel attacks are a considerable threat to implementations of classic public-key schemes. For lattice-based cryptography, however, this class of attacks is much less understood, and only a small number of previous works show attacks. Primas et al., for instance, present a single-trace attack on the Number Theoretic Transform (NTT), which is at the heart of many efficient lattice-based schemes.
They, however, attack a variable-time implementation and also require a rather powerful side-channel adversary capable of creating close to a million multivariate templates. Thus, it was an open question if such an attack can be made practical while also targeting state-of-the-art constant-time implementations.
In this paper, we answer this question positively. First, we introduce several improvements to the usage of belief propagation, which underlies the attack. And second, we change the target to encryption instead of decryption; this limits attacks to the recovery of the transmitted symmetric key, but in turn, increases attack performance. All this then allows successful attacks even when switching to univariate Hamming-weight templates.
We evaluate the performance and noise resistance of our attack using simulations, but also target a real device. Concretely, we successfully attack an assembly-optimized constant-time Kyber implementation running on an ARM Cortex M4 microcontroller while requiring the construction of only 213 templates.

Keywords: side-channel attacks · post-quantum cryptography · lattice-based cryptography · belief propagation

1 Introduction

For implementations of classic public-key schemes, such as RSA, DH, and ECC, there exist many side-channel attacks capable of retrieving the secret key using just a single execution trace. Ranging from the textbook example of distinguishing squarings and multiplications with the naked eye to more sophisticated methods, such as horizontal collision correlation [9] capable of attacking even constant-time implementations, such single-trace attacks can be considered to be a prime threat and are able to bypass many randomization countermeasures.

For lattice-based cryptography, the largest family of schemes currently running in the second round of the NIST post-quantum standardization process [21],

the sincerity of this threat is much less clear. Compared to most classic constructions, it is relatively straightforward to make implementations of lattice-based cryptography both constant time[1] and free of secret-dependent memory accesses, thereby eliminating the most glaring side-channel leaks. Possibly due to this reason, many previous works on secure implementations of lattice-based cryptography primarily focused on masking and thus differential side-channel attacks [25,24,22,8].

Still, single-trace attacks have received some prior attention and were shown to be possible. In particular, Primas et al. [23] proposed an attack on the Number Theoretic Transform (NTT), which is an integral part of efficient implementations of many lattice-based schemes, e.g., NewHope [2] and Kyber [6]. Simply speaking, the NTT resembles a Fast Fourier Transform (FFT), but works over $\mathbb{Z}_q$ instead of over the complex plane. It enables fast multiplication of polynomials in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, which are a common sight in schemes based on structured lattices. The attacks of Primas et al. follow the path of soft-analytical side-channel attacks [28]. That is, they first perform a side-channel template matching [12] on certain intermediates, construct a graph describing the NTT and all of its intermediate computations, include the observed leakage information in this graph, and finally run a message-passing algorithm known as belief propagation. The recovered secret input is either the key itself or can be used to recover said key.

However, while this attack can even bypass certain countermeasures, it does leave open the question of practicality. For their evaluations on a real device, they build close to one million templates. Besides, they attack a variable-time implementation. While the attack, as such, does not require timing differences, it does benefit from them. And finally, they mainly focus on attacking the decryption process. While the most apparent target, the involvement of long-term secrets makes the need for side-channel protections obvious. Encryption, however, only deals with ephemeral secrets and might thus see less care in side-channel protections. Still, a successful attack on encryption can lead to a compromise of the entire system.

Our Contribution. In this paper, we address the above limitations and show that single-trace attacks on the NTT can be made truly practical. Several improvements to the attack, alongside the choice of a different target, allow us to attack a constant-time microcontroller implementation of the Kyber lattice-based key exchange [6], all while requiring only 213 univariate Hamming-weight templates.

More concretely, we include three improvements to belief propagation in the context of side-channel attacks on the NTT. We merge certain nodes in the graph representation of the NTT, make use of message damping, and introduce a new message schedule. These changes lead to higher accuracy of computed marginal probabilities and thus to better attack performance. The runtime of belief propagation, while increased, still stays very reasonable.

[1] At least when using a simple error distribution, such as centered binomials.

As already hinted above, we change the concrete target of our attack. Primas et al. attacked the inverse NTT transformation during decryption. Decryption involves the private key, which makes not only attacks worthwhile, but also the need for careful side-channel protection obvious. We target encryption instead. While this limits attacks to recovering the exchanged symmetric keys, it focuses on a part seemingly requiring less side-channel protection. Also, in encryption, the inputs of the NTT are confined to a narrow interval, which further aids attack performance.

These changes and performance improvements allow a simplification of the physical part of the attack. That is, by switching to Hamming-weight templates and targeting load/store operations instead of multiplications, the number of required templates and thus also traces for template building is cut down drastically.

We evaluate our attack for different noise levels using simulations. Furthermore, we study the effects of masking and recent implementation techniques, such as lazy reduction, on the attack performance. Finally, we demonstrate the attack using real power measurements of an STM32F4 microcontroller running a constant-time ASM-optimized Kyber implementation. Using just 213 univariate Hamming-weight templates, the entire secret NTT input can be recovered with a probability of up to 95%. Finally, we note that our attacks can be easily ported to many other implementations that make use of the NTT.

Outline. In Section 2, we briefly describe the concrete target of our attacks, namely the Kyber key exchange, and discuss its implementation aspects. In Section 3, we present details of the attack by Primas et al. [23] and also discuss its shortcomings. After having covered the necessary background, we show all our improvements and adaptations in Section 4. We then evaluate the attack using simulations in Section 5 and target a real device in Section 6. In Section 7, we discuss the applicability and effectiveness of previously proposed countermeasures.

2 Lattice-Based Cryptography

In this section, we briefly recall the lattice-based key-exchange Kyber. We also describe efficient implementation techniques, both for Kyber and lattice-based cryptography in general.

2.1 Kyber

The Kyber key exchange [6] is currently running in the second round of the NIST standardization process [21]. In its core, Kyber resembles the Ring-LWE encryption scheme proposed by Lyubashevsky, Peikert, and Regev [18], but it bases its security on the Module Learning-With-Errors assumption (MLWE) [16] instead of Ring-LWE. This means that it operates with matrices/vectors containing polynomials defined over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. We use boldface letters to differentiate matrices/vectors of polynomials from single polynomials.

Already in its specification, Kyber prescribes usage of the NTT for efficient polynomial multiplication. Via point-wise multiplication of transformed polynomials, i.e., $ab = \mathsf{NTT}^{-1}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$, multiplication can be performed in time $\mathcal{O}(n \log n)$. We use $\hat{a}$ as shorthand for the NTT-transformed of a, with $\hat{\mathbf{a}}$ we denote vectors where all component polynomials are transformed.

The core public-key encryption scheme (PKE) only offers IND-CPA security. For this reason, the Kyber authors apply a variant of the Fujisaki-Okamoto transform [14] to build an IND-CCA2 secure key-encapsulation mechanism (KEM). In essence, the transform requires a re-encryption of the decrypted message using the randomness seed used for the original encryption, which is embedded in the ciphertext. Only if the recomputed and the received ciphertexts match, the decrypted message is released. Since recovering the key/message used in the underlying PKE directly leads to key/message recovery of the KEM, we omit details of the transform and only focus on the PKE. We further omit aspects regarding, e.g., efficient packing, and give a simplified but conceptually identical description. For further details, we refer to the Kyber specification [6].

Algorithm 1 gives the key-generation procedure. The function $\mathsf{Sample}_\mathsf{U}$ samples the $(k \times k)$-matrix $\hat{\mathbf{A}}$ from uniform using the seed ρ, which is also part of the public key. The sampling is performed directly in the NTT domain. Then, the coefficients of $\mathbf{s}$ and $\mathbf{e}$ are sampled following a centered binomial distribution with support $[-\eta, \eta]$ using $\mathsf{Sample}_\mathsf{B}$ with seed σ. Afterward, the NTT is applied to all component polynomials of $\mathbf{s}$ independently to receive the secret key $\hat{\mathbf{s}}$. The result $\mathbf{t} \coloneqq \mathbf{A}\mathbf{s} + \mathbf{e}$ is the public key.

Algorithm 1 Kyber-PKE Key Generation (simplified)

Output: Public key pk, private key sk

1: Choose uniform seeds ρ, σ
2: $\hat{\mathbf{A}} \in R_q^{k \times k} \coloneqq \mathsf{Sample}_\mathsf{U}(\rho)$ ▷ Generate uniform $\hat{\mathbf{A}}$ in NTT domain
3: $\mathbf{s} \in R_q^k \coloneqq \mathsf{Sample}_\mathsf{B}(\sigma||0)$ ▷ Sample private key $\mathbf{s}$ (binomial distribution)
4: $\mathbf{e} \in R_q^k \coloneqq \mathsf{Sample}_\mathsf{B}(\sigma||1)$ ▷ Sample error $\mathbf{e}$ (binomial distribution)
5: $\hat{\mathbf{s}} \coloneqq \mathsf{NTT}(\mathbf{s})$ ▷ NTT for efficient multiplication
6: $\mathbf{t} \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s}}) + \mathbf{e}$ ▷ $\mathbf{t} \coloneqq \mathbf{A}\mathbf{s} + \mathbf{e}$
7: **return** $(pk \coloneqq (\mathbf{t}, \rho), sk \coloneqq \hat{\mathbf{s}})$

Encryption is shown in Algorithm 2. After recomputation of $\hat{\mathbf{A}}$ from the seed ρ, the variables $\mathbf{r}, \mathbf{e}_1, e_2$ are sampled. The seed τ used for this sampling is made explicit to allow the re-encryption required for the CCA2 transform. The ciphertext c consists of two parts, where the second component c_2 contains m encoded as an element in $\mathcal{R}_q$. The decryption process (Algorithm 3) requires to recover this m from a noisy version.

The Kyber authors originally specified three parameter sets. In this paper, we primarily focus on the original Kyber768 set given by $(n = 256, k = 3, q = 7681, \eta = 4)$. Kyber512 and Kyber1024 mainly differ in the used k. Since we will target individual NTT executions, k does not impact attack performance,

Algorithm 2 Kyber-PKE Encryption (simplified)

Input: Public key $pk = (\mathbf{t}, \rho)$, 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: $\mathbf{r} \in R_q^k \coloneqq \mathsf{Sample}_{\mathsf{B}}(\tau||0)$
3: $\mathbf{e}_1 \in R_q^k \coloneqq \mathsf{Sample}_{\mathsf{B}}(\tau||1)$ ▷ Sample noise $\mathbf{r}, \mathbf{e}_1, e_2$
4: $e_2 \in R_q \coloneqq \mathsf{Sample}_{\mathsf{B}}(\tau||2)$
5: $\hat{\mathbf{r}} \coloneqq \mathsf{NTT}(\mathbf{r})$ ▷ NTT for efficient multiplication
6: $\mathbf{c}_1 \coloneqq \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{c}_1 \coloneqq \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
7: $c_2 \coloneqq \mathsf{NTT}^{-1}(\mathsf{NTT}(\mathbf{t})^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Encode}(m)$ ▷ $c_2 \coloneqq \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Encode}(m)$
8: **return** $c \coloneqq (\mathbf{c}_1, c_2)$

Algorithm 3 Kyber-PKE Decryption (simplified)

Input: Public key $pk = (\mathbf{t}, \rho)$, secret key $sk = \hat{\mathbf{s}}$, ciphertext $c = (\mathbf{c}_1, c_2)$
Output: Message m

1: $m \coloneqq \mathsf{Decode}(c_2 - \mathsf{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{c}_1)))$ ▷ $m \coloneqq \mathsf{Decode}(c_2 - \mathbf{s}^T\mathbf{c}_1)$
2: **return** m

at least as long the success probability on single NTTs is close to 1. We note that very recently, the Kyber parameters were tweaked for round 2 of the NIST standardization process. All parameter sets now feature $(q = 3329, \eta = 2)$.[2] We will later show that this change is beneficial to our attack. Note that we will always use the original parameter set unless stated otherwise.

2.2 Efficient and Secure Implementation

The rising popularity of lattice-based cryptography in the last decade has also led to many efficient constant-time implementation techniques. We now give details to the two most relevant for this work.

Number Theoretic Transform (NTT). As already stated above, the NTT allows efficient multiplication in $\mathcal{R}_q$ by pointwise multiplying two forward transformed polynomials and transforming the result back[3]. In essence, the NTT is a Discrete Fourier Transform (DFT) over a prime field $\mathbb{Z}_q$ and can thus be implemented using the same techniques as found in the Fast Fourier Transform (FFT). As shown in Fig. 1 with the example of a 4-coefficient NTT, the transformation consists of chaining $\log_2 n$ layers of so-called butterflies. A butterfly takes two inputs, one of them is multiplied with a certain power of ω_n, where ω_n is a primitive n-th root of unity in $\mathbb{Z}_q$. The product is then both added to (upper branch) and subtracted from (lower branch) the second input.

[2] The new parameter set also requires some minor modifications to the NTT, such as different constants and omission of the last butterfly layer.
[3] Here, the NTT is defined to include the scaling required to compute products in $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ instead of $\mathbb{Z}_q[x]/\langle x^n - 1\rangle$. We point to [6] for further details.

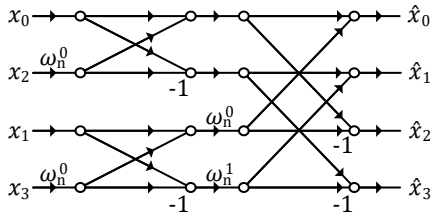


Fig. 1. 4-coefficient NTT

Constant-time and efficient reductions. Early implementations of the NTT often used straight-forward and variable-time modular reduction techniques. For instance, de Clercq et al. [13] use ARMs conditional operations for reductions after additions and subtractions, as well as integer divisions for reductions after multiplications. On most embedded devices, such divisions do not run in constant time.

More recent implementations, e.g., the Cortex-M optimized NewHope implementation by Alkim, Jakubeit, and Schwabe [4], frequently make use of constant-time variants of the established Montgomery and Barret reduction techniques. Constant-time is reached by omitting the final conditional subtractions. In other words, the result is not always reduced back to $[0, q-1]$, but can be larger. This can also be used for efficiency gains by, e.g., skipping reductions after additions (*lazy reduction*).

2.3 Protected Implementations

Constant-time operations mitigate timing attacks, both on small devices such as microcontrollers and large ones like PCs. Protecting against other types of side-channel attacks, e.g., Differential Power Analysis (DPA), requires more effort. There do already exist works addressing this issue and proposing DPA-secured implementations of lattice-based cryptography; they use masking as their main protection mechanism [25,22,8].

Since the NTT is a linear transformation, it is trivial to mask. When s is the sensitive input, then one can sample a uniformly random masking polynomial $m \in \mathcal{R}_q$, compute the NTT on m and $(s-m)$ independently, and finally add the shares back again if needed. Masking the sampling of error polynomials and the decoding of the noisy message in decryption is much more intricate. Since we do not attack these operations, we refer to Oder et al. [22] for details.

Oder et al. [22] further employ hiding techniques. They shuffle the ordering of linear operations, such as the pointwise multiplication, and blind polynomials with a random scalar. The latter method was first introduced by Saarinen [26].

3 Single-Trace Attacks on Lattice-Based Cryptography

Masking is very efficient in protecting against DPA-like attacks. Still, single-trace attacks are potentially able to bypass masking as well as other defenses. There do already exist earlier works showing the feasibility of such attacks in the context of lattice-based cryptography. Recently, horizontal side-channel attacks on matrix-vector multiplications found in schemes over unstructured lattices, such as Frodo [3], were demonstrated [7,11]. These attacks, however, do not carry over to schemes using structured lattices, as they typically use faster multiplication methods such as the NTT or Karatsuba's method.[4]

Primas et al. [23] propose a single-trace attack on the NTT. Since this attack is the basis of our work, we now introduce the required background. First, we recall soft-analytical side-channel attacks and the belief propagation algorithm, which form the basis of the attack. Then, we will present the attack details and discuss some of its shortcomings.

3.1 Soft-Analytical Side-Channel Attacks

The attack of Primas et al. is an instance of a soft-analytical side-channel attack (SASCA), which were first proposed by Veyrat-Charvillon et al. [28]. In SASCA, one first performs a side-channel template attack [12] on certain intermediates and retrieves probabilities conditioned on the leakage. In other words, one gets $\Pr(T = t|\ell)$, where T is an attacked intermediate, t runs through all of the possible values of T, and ℓ is the observed side-channel leakage. Then, one constructs a factor graph modeling the attacked algorithm and its specific implementation. After including the conditioned probabilities into this graph, the belief-propagation algorithm is run, which returns marginal probabilities for all inputs (including the involved key components), outputs, and intermediates processed by the algorithm. We now give a more thorough explanation.

Belief propagation. We base our descriptions of belief propagation (BP) on MacKay [19, Chapter 26], as also done by previous works using SASCA [28,23].

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 (FG). 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

[4] Aysu et al. [7] do also run their attack for the RLWE-based scheme NewHope [2]. However, their attacked implementation uses schoolbook multiplication instead of the NTT, resulting in a drastically increased runtime.

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:

$$u_{n\to m}(x_n) \;=\; \prod_{m'\in\mathcal{M}(n)\setminus\{m\}} v_{m'\to n}(x_n), \tag{1}$$

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

2) from factor to variable:

$$v_{m\to n}(x_n) \;=\; \sum_{x_m\setminus n}\left(f_m(\mathbf{x}_m)\prod_{n'\in\mathcal{N}(m)\setminus m} u_{n'\to m}(x'_n)\right), \tag{2}$$

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 $Z = \sum_{x_n} Z_n(x_n)$.

BP is guaranteed to return the correct marginals only if the factor graph is acyclic. If this is not the case, then the same update rules can still be used in what is then called loopy BP. This variant might not even converge, but when it does, it often gives sufficiently precise approximations to the true marginals. The performance of loopy BP, i.e., the quality of the approximations and converge properties, is inversely proportional to the length of the loops. Put simply, loops in the factor graph introduce positive feedback, which can cause overconfidence in certain beliefs and subsequently even oscillations, especially when deterministic factors are involved. Longer loops are less susceptible to this effect. Note that there do exist approaches, such as generalized belief propagation [30], aiming at significantly improving the quality of the marginal probabilities in loopy graphs. They, however, can come with significantly increased computational runtime.

3.2 Single-Trace Attacks on the NTT

By running a soft-analytical side-channel attack, Primas et al. [23] can recover the secret NTT inputs after observing just a single trace. Concretely, they recover the inputs of the inverse NTT in decryption (Algorithm 3), and can then derive the key $\mathbf{s}$.[5] The NTT appears to be a fitting target for SASCA since each stored intermediate is computed using relatively simple combinations (additions and subtractions) of just two intermediates of the previous NTT layer.

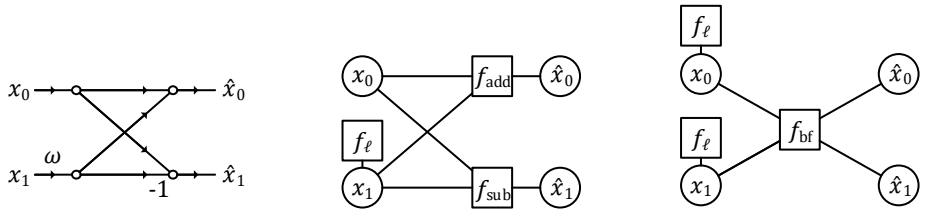
[5] They target the original LPR scheme, which has very similar encryption and decryption routines.

Fig. 2 demonstrates how one can construct a factor graph for the NTT. Fig. 2a shows a single butterfly for reference. Note that since such a butterfly is equivalent to a length-2 NTT, we denote the outputs as $\hat{x}_0$ and $\hat{x}_1$. Fig. 2b then depicts the corresponding factor graph as constructed by Primas et al. In this graph, variable nodes and factor nodes are represented by circles and squares, respectively. The factor nodes can be further split into two groups. Factor f_ℓ models the observed side-channel information, i.e., the outcome of the template matching. More concretely, we have $f_\ell(i) = \Pr(x = i|\ell)$, where x is the matched intermediate. Primas et al. perform template matching on the modular multiplication with ω, which is why they receive information on x_1.

The second group of factors, consisting of f_{add} and f_{sub}, then model the deterministic relationships between the variable nodes as specified by the NTT. For, e.g., the addition in the upper branch, we get:

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

Due to the deterministic nature of f_{add} and f_{sub}, the factor-to-variable update rule stated in Eq. 2 can be computed in time $\mathcal{O}(q^2)$ by simply enumerating all q^2 possible input combinations. Primas et al. can decrease the runtime to $\mathcal{O}(q \log q)$ by using cyclic properties of modular addition and FFTs of length q.



(a) Single butterfly (b) FG of [23] (c) Our FG

Fig. 2. Comparison of a single butterfly with possible factor-graph representations

Shortcomings. While Primas et al. demonstrate the possibility of side-channel attacks on the NTT, their attack falls somewhat short of being fully practical.

They perform template matching on modular multiplications. This requires constructing templates for all possible combinations of x_1 and ω taking q and $n/2$ possible values, respectively. For their evaluated parameter set, also featuring $(n = 256, q = 7681)$, close to a million templates are required. Each template is constructed using 100 traces, thus summing up to 100 million traces used for evaluation. They also assume time-invariance of leakage, which allows them to condense analysis to just that of butterflies without regarding its position in the trace. We found that this assumption might not always hold (cf. Section 6.2).

In addition, they attack the variable-time NTT implementation by de Clercq et al. [13], which makes use of ARMs conditional instructions and variable-time integer division. Note that the attack as such does not require timing leakage, it can easily be adapted to constant-time implementations. However, the inclusion of timing information is beneficial to attack performance. Thus, the applicability to constant-time implementations is unknown.

Finally, we note that the butterfly factor graph shown in Fig. 2b contains short loops, which is detrimental to the performance of BP. In fact, due to the bipartite and singly-connected nature of factor graphs, no shorter loops are possible in any such graph.

4 Reaching Practical Single-Trace Attacks

We now address these problems and show how single-trace attacks on the NTT can be made truly practical, even on constant-time implementations. First, in Section 4.1, we decrease the number of required templates. In combination with the lack of timing information, the attack now fails. For this reason, we adopt several improvements to the belief-propagation algorithm for our scenario, as explained in Section 4.2. Then, in Section 4.3, we explain why attacking encryption instead of decryption can boost performance even further.

4.1 Decreasing the Number of Templates

Our method to decrease the number of required templates and thus also traces for template building to a more considerate amount is relatively simple. Instead of performing a template matching on modular multiplication and constructing templates for each possible input combination, we target loading/storing of butterfly inputs/outputs to and from RAM (cf. Fig. 2c using leakage of just loading inputs). In addition, we do not construct templates for every single possible value, but only for Hamming weights. Under ideal circumstances, this limits the number of templates to just $\lceil \log_2 q \rceil + 1 = 14$, which, compared to the previous attack, is a reduction by a factor of over 70 000. Apart from this reduction, (univariate) Hamming-weight templates are also significantly easier to port from one device to another (compared to multivariate value-based templates).

As it turns out, however, the information loss due to switching to such simpler templates and additionally losing all timing leakage–we only target constant-time implementations–is too high. The attack fails, even for the noise-free case. For this reason, we will now propose improvements to the attack, which allow successful message recovery for such a more constrained attacker.

4.2 Improving Belief Propagation for the NTT

There already exists a large body of work studying ways to improve the performance of belief propagation in cyclic factor graphs. We adopted three concrete methods for use with the NTT and will now describe them in depth.

Butterfly factors. The factor graph shown in Fig. 2b contains very short loops, which, especially in conjunction with deterministic factors, can lead to convergence problems and overall bad performance of loopy BP [27]. Such network configurations and the resulting problems are however not exclusive to the NTT. As shown by Storkey [27] and Yedidia [29], similar problems also appear when applying BP to other FFT-like networks. Storkey analyzes BP in context of ordinary real-valued FFTs, whereas Yedidia focuses on Reed-Solomon codes, which can be represented by NTTs/FFTs over GF(q).

To increase BP performance, both Stork and Yedidia propose to cluster the factors belonging to the same butterfly and thereby enforce all of its input/output relations at once. We follow their approach and replace factors f_{add}, f_{sub} with a single *butterfly factor* f_{bf}. As seen in Fig. 2c, this eliminates the loop inside each butterfly. The full factor graph of the NTT, built by connecting multiple instances of the butterfly FG (cf. Fig. 1), will still contain loops. These loops, however, are longer, which will lead to increased performance. Yedidia notes that this clustering constitutes a simple form of generalized belief propagation [29,30].

Butterfly factors are specified as:

$$f_{\text{bf}}(x_0, x_1, \hat{x}_0, \hat{x}_1) = \begin{cases} 1 & \text{if } x_0 + x_1\omega = \hat{x}_1 \bmod q \text{ and } x_0 - x_1\omega = \hat{x}_1 \bmod q \\ 0 & \text{otherwise} \end{cases}$$

With the increased accuracy, however, comes also an increase in computational runtime. By making use of an FFT of length q, the update rules for f_{add} and f_{sub} can be evaluated in time $\mathcal{O}(q \log q)$. This does not carry over to f_{bf}. Instead, all input combinations need to be enumerated, thereby increasing the runtime to $\mathcal{O}(q^2)$.

For typical parameters of lattice-based encryption, with $q \approx 2^{13} - 2^{14}$ [6,2], this is still very much practical, as will later be demonstrated. For the moduli used by lattice-based signatures, e.g., the schemes Dilithium [17] and qTesla [10] both use $q \approx 2^{23}$, practicality cannot be claimed anymore. When also considering that the NTT in, e.g., Dilitihium, consists of 2^{10} butterfly invocations, then it becomes clear that the previous method with split factors needs to be used there.

Optimized message schedule. Another property that can influence convergence and accuracy in loopy BP is the chosen message schedule, i.e., the order in which messages are computed and passed between nodes. The most straightforward schedule is to update all variable or factor nodes simultaneously. This approach is followed by Primas et al. [23]. When using the original representation of Fig. 2b, then completing a loop requires just 2 iterations of evaluating Equations (1) and (2) (for our proposed representation, this number increases to 4). It, however, can take up to $2 \log_2 n$ iterations for any two nodes in the full NTT graph to communicate.

This is clearly not ideal, which is why we adopt the schedule also used by Storkey [27]. That is, we first pass messages from the NTT input to the output (layer by layer), and then back again. This does not affect the number of iterations required for completing a loop but allows any two nodes in the factor

graph to communicate in just a single iteration. This becomes especially advantageous when changing the target to encryption, which features inputs with small support.

Message damping. While the above two methods greatly increase accuracy, convergence is still not guaranteed. For this reason, we finally also adopt message damping. It aims to dampen oscillations by computing a weighted average of the new and the previous message. When denoting α as the damping factor and $u_{n\to m}^{\text{prev}}$ as the message sent from node to factor in the previous iteration, then the dampened version of (1) is:

$$u_{n\to m}(x_n) \;=\; \alpha \left(\prod_{m' \in \mathcal{M}(n)\setminus\{m\}} v_{m'\to n}(x_n) \right) + (1-\alpha) u_{n\to m}^{\text{prev}}(x_n) \qquad (3)$$

For all of our later experiments, we set the damping factor α to 0.9.

4.3 Changing Targets

The decryption process described in Algorithm 3 involves the secret key $\mathbf{s}$ and is thus the obvious first target of a side-channel attack. We now argue that encryption, while not involving $\mathbf{s}$, can also be a very interesting target and is significantly easier to attack.

First, encryption only deals with ephemeral secrets. While this means that the side-channel attack has to be performed for each individual message, the lack of long-term secrets makes it very tempting to use implementations devoting fewer resources for side-channel protections, or maybe even an unprotected implementation. We want to prove this intuition wrong.

Second, the Fujisaki-Okamoto CCA2-transform employed by Kyber and many other lattice-based KEMs requires a re-encryption of the message. The message is then only released if the recomputed ciphertext matches the received one. This means that attacks are not restricted to sending devices, but can also be mounted on the receiving end, i.e., on devices having access to the secret key.

Third, encryption involves an NTT with inputs over a very narrow support: error polynomials follow a centered binomial distribution over $[-\eta, \eta]$, with $\eta = 4$ in our analyzed parameter set. The inputs to the inverse NTT in decryption, however, can be considered uniform in $\mathbb{Z}_q$. Information on the narrow support can easily be integrated into the factor graph; it suffices to apply Bayes' theorem to the leakage-likelihood obtained in the input layer and use the result in the factor nodes f_ℓ. The first iteration of our forward-backward message passing then immediately spreads the information to all later layers and ensures, e.g., that in the second layer, only $(2\eta+1)^2$ values have a non-zero probability. The narrow support allows BP to pick the correct value under much more noise, as we will later demonstrate.

Fourth and finally, we note that key generation (Algorithm 1) also features an NTT with small inputs, namely that of the private key $\mathbf{s}$. Thus, our attack

also applies here. This becomes interesting on devices which either use the plain IND-CPA secure scheme described in Section 2.1 and thus only use ephemeral keys, or only store the seed σ and regenerate $\mathbf{s}$ each time to save space in secure non-volatile storage.

Attacking encryption. For the actual attack on encryption, we target the forward NTT of $\mathbf{r}$, found in line 5 of Algorithm 2. We picked this invocation of the NTT, since all others in Algorithm 2 work on polynomials which are uniform over $\mathbb{Z}_q$. Since $\mathbf{r} \in \mathcal{R}_q^k$ is comprised of k polynomials, we have to attack all k independent invocations of the NTT. Then, we can compute the message m by using line 7. That is, $m = \mathsf{Decode}(c_2 - \mathbf{t}^T\mathbf{r}) = \mathsf{Decode}(e_2 + \mathsf{Encode}(m))$.

5 Evaluation and Additional Scenarios

After having described our attack in depth, we now evaluate it using leakage simulations and additionally explore more attack scenarios. First, in Section 5.1, we analyze the improvements proposed for BP in the context of the NTT. Then, we focus on implementations employing the masking countermeasure in Section 5.2. In Section 5.3, we finally study the effects of state-of-the-art implementation techniques.

Leakage simulation. For all our simulations, we make use of the established *Hamming weight with additive Gaussian noise* model. Thus, when processing a value x, we receive simulated leakage $\ell = \mathsf{HW}(x) + \mathcal{N}(0, \sigma_{\mathsf{HW}})$. Here, HW denotes the Hamming-weight function, and $\mathcal{N}(0, \sigma_{\mathsf{HW}})$ describes a random sample from the normal distribution with zero mean and standard deviation σ_{HW}.

For each simulation run, we generate one such sample for each intermediate state variable. Thus, we have n samples in each layer (including inputs and outputs), this sums up to $n(\log_2(n) + 1) = 2\,304$ samples. Note that in practice, intermediates in inner layers of the NTT would leak twice: once during storing, once during loading. For the sake of simplicity, we only generate a single sample here. Finally, we perform a template matching on all generated samples and retrieve the corresponding conditioned probabilities $\Pr(X = x|\ell)$.

Attack implementation. We implemented our evaluations and attacks, including the belief-propagation algorithm, in Matlab. The most time-critical component, namely the factor-to-variable update of butterfly factors f_{bf}, was outsourced to multi-threaded C++ code. We performed up to 50 full message-passing iterations but abort as soon as convergence is reached for all variable nodes in the network. All experiments were run on an Intel Xeon E5-4669 v4 (2.2 GHz).

Evaluation of attack performance is done for varying values of σ_{HW}. For each scenario and analyzed noise level, we performed at least 100 experiments. We computed the success rate by counting the experiments where belief propagation correctly classifies all n NTT inputs, i.e., assigns the highest probability to the correct values.

5.1 Evaluating Improvements to BP

For our first evaluations, we analyze the effects of the improvements proposed in Section 4.2: the introduction of butterfly factor nodes, a changed message schedule, and the use of damping. This evaluation also doubles as a general analysis of the noise resistance of our attack.

We target a generic constant-time but otherwise unprotected implementation of the Kyber-NTT. The exact internal operations of this implementation are not important, at least as long the factor graph model and the actual implementation are consistent regarding the attacked intermediates. As we target the loads and stores at the inputs and outputs of butterflies, the exact methods used for, e.g., modular multiplication, are mostly irrelevant.

Fig. 3 shows the outcome, for both the original ($q = 7681, \eta = 4$) and the tweaked ($q = 3329, \eta = 2$) Kyber parameter sets. Without the improvements, but still using the same load/store leakage, a success rate of more than 0.9 can be maintained up to and including $\sigma_{\mathsf{HW}} = 0.9$. With our changes, this threshold is increased to $\sigma_{\mathsf{HW}} = 1.5$. When computing an SNR and thus looking at the variance σ^2_{HW}, then this difference corresponds to almost a tripling of the acceptable noise level. The smaller values used by the new parameter set lead to a slight improvement in the success rate.

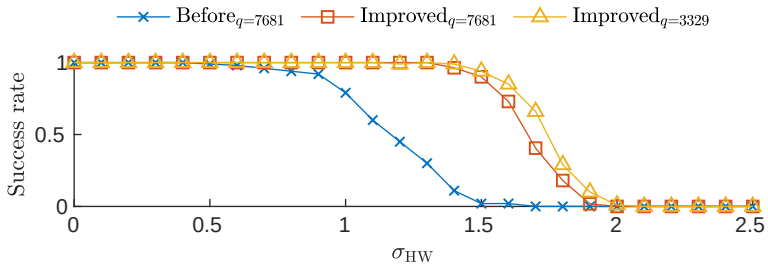


Fig. 3. Comparison of the attack success rate, with and without the optimizations proposed in Section 4.2. The improved version was evaluated for both the original ($q = 7681$) and tweaked ($q = 3329$) parameter sets.

On a single core of our system, the runtime of a full forward-backward iteration of belief propagation using $q = 7681$ is roughly 2 minutes. In low-noise cases, 2 such iterations are already sufficient. For $\sigma_{\mathsf{HW}} = 1.5$, the average number of iterations (for convergent experiments) rises to 9. For this noise level, all failed experiments correspond to non-convergence of BP.

5.2 The Case of Masking

As noted in Section 2.3, masking the NTT is straightforward. So is, at least in theory, the adaptation of a single-trace attack to the masked case. One can

simply recover each share individually and add them up to receive the unmasked input. This approach is used by Primas et al. [23]. In their evaluations, masking alone does not significantly decrease the attack success rate. In our scenario, this is no longer the case. We specifically target the NTT of $\mathbf{r}$ due to the small input coefficients. When using masking, this advantage is lost, as all input coefficients become uniformly distributed over $\mathbb{Z}_q$.

We reintroduce the information on the narrow support as follows. Instead of running belief propagation on two factor graphs corresponding to the two shares individually, we adjoin graphs at the input layer using factor nodes f_{bino}. These nodes ensure that the sum of the two inputs is consistent with the centered binomial distribution over $[-\eta, \eta]$. When using $\mathcal{B}_\eta$ to denote the density of said distribution, and x', x'' as the two shares of the input, then we can write $f_{\text{bino}}(x', x'') = \mathcal{B}_\eta(x' + x'' \bmod q)$.

Fig. 4 shows the simulation results. Running BP on the shares independently does not yield satisfactory results, even for perfect Hamming-weight leakage. The introduction of f_{bino} brings the success rate close to 1, at least when $\sigma_{\mathsf{HW}} \leq 0.3$. While this is a drastic reduction compared to the unprotected case, it at least allows attacks in low noise scenarios. Here, also note that our attacker only uses Hamming-weight templates. A more powerful adversary might attack masked implementations even in high-noise settings.

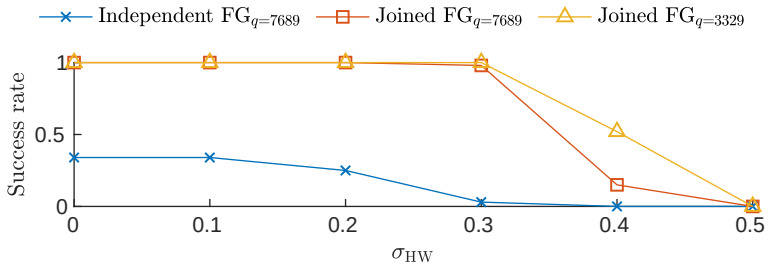


Fig. 4. Success rate of our attack on a masked implementation, both using independent factor graphs and joined graphs

5.3 The Case of Lazy Reductions

Up until now, we analyzed a relatively generic implementation of the NTT. There, operands are always in the range $[0, q-1]$. As mentioned in Section 2.2, this is not true for many recent implementations. They make use of constant-time variants of the Montgomery and Barret reduction, both of which do not necessarily reduce operands down to the base range. Instead, they reduce to a representative (of the equivalence class) only guaranteed to be smaller than, e.g., 2^{16}. Also, reductions after, e.g., additions, can be skipped for performance improvements (lazy reduction).

Such an implementation could, at least theoretically, be attacked using the same factor-graph representation. After performing the template matching on the now larger range of possible values, one could compute the probability of each equivalence class by summing up the probability of each possible representative. Due to just using Hamming-weight leakages, we do not think that such an approach is fruitful.

Instead, we modify the graph to directly model the changed operations. Concretely, we target the assembly-optimized ARM Cortex M4 implementation of Kyber provided by the PQM4 library [15].[6] It uses Montgomery reductions after modular multiplications and Barret reductions after additions and subtractions. Reductions after additions are skipped in each other layer. We integrate all that in the butterfly factors. When denoting MRed and BRed as the used reduction routines, and looking at a layer where no reduction is skipped, then we have

$$f_{\mathrm{bf}}(x_0, x_1, \hat{x}_0, \hat{x}_1) = \begin{cases} 1 & \text{if } \mathsf{BRed}(x_0 + \mathsf{MRed}(x_1\omega)) = \hat{x}_0 \text{ and} \\ & \quad \mathsf{BRed}(x_0 + 4q - \mathsf{MRed}(x_1\omega)) = \hat{x}_1 \\ 0 & \text{otherwise} \end{cases}$$

The results of the simulations using this model are shown in Fig. 5. The analyzed implementation only supports the original parameter set with $q = 7681$, which is why we limit analysis to this scenario. Note, however, that our earlier results indicate better attack performance for the tweaked parameter set. Compared to our earlier results, the 90% success-rate threshold is now decreased to $\sigma_{\mathsf{HW}} = 1.3$. As an effect of the larger input ranges, the single-core runtime of a full iteration increases to approximately 8 minutes.

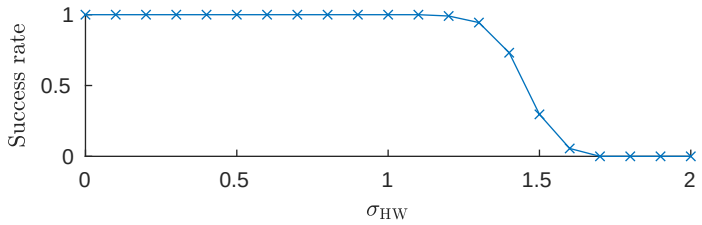


Fig. 5. Success rate of our attack when modeling constant-time and lazy reductions

6 Attacking a Real Device

The previous section already analyzed an optimized microcontroller implementation, but still resorts to leakage simulations. We now show that our attack carries over to an actual device.

[6] Shortly after the initial publication of this paper, the Kyber implementation in PQM4 was updated. For reference, we used the version found at `https://github.com/mupq/pqm4/releases/tag/Round1`.

6.1 Measurement Setup

The ARM Cortex M4 appears to be the standard target for embedded software implementations of post-quantum cryptography [5]. For this reason, we also performed our side-channel analysis on such a device. More concretely, we performed power measurements of an STM32F405 microcontroller atop the STM32F4 target for the ChipWhisperer UFO board [20]. The power consumption was measured using an AD8129A differential amplifier across an onboard shunt resistor. The 8 MHz device clock was externally generated using a function generator. This was done to simplify synchronization across traces.

This device then ran the optimized Kyber implementation of the PQM4 library [15], which we already targeted in the previous section. More correctly, we only run the forward NTT of error polynomials. Each trace captures an entire NTT execution; we used a dedicated trigger pin to signal the start and end of this operation. We recorded 2 000 traces and then split this set into 1 900 templating traces and 100 attack traces. Note that, while both sets of traces were recorded on the same device, the use of simple univariate Hamming-weight templates makes porting of templates to similar but different devices much more realistic compared to the previous attack.

6.2 Trace Analysis and Attack

As we want to demonstrate practicality, we keep the trace analysis relatively simple by using univariate Hamming-weight templates. For determining the position of leaking operations, we run a correlation analysis along the entire length of the traces, for each of the 2 304 attacked intermediates. The single position of the highest peak (in absolute value) was then selected as point of interest.

The switch to Hamming-weight templates would ideally allow a reduction to just 14 templates. However, we found that the power leakage does show a certain degree of time dependence. Even after basic trace normalization, such as point-wise subtraction of the mean and normalization to a standard deviation of 1, the same operation showed slightly different behavior when executed at a different point in time. We suspect the different flow of instructions preceding our attacked operations, alongside the low-pass behavior of the power network, to be the cause.

Construction of templates for each position is still not required, as we follow an intermediate approach. We build two sets of templates for each NTT layer. The first set targets the upper branch in all butterflies of the respective layer, the second set targets the lower branch (cf. Fig. 2a). Each template is thus used $n/2 = 128$ times. As not all Hamming weights are possible in all layers, one has to build a total of 213 templates.

After matching these templates on an attacked trace, we use the factor-graph representation already established in Section 5.3. Out of our 100 performed experiments, 83 yield the correct NTT input. Since we need to attack $k = 3$ independent NTTs, the total success rate can be estimated to be $0.83^3 \approx 0.57$.

6.3 Increasing the Success Rate

The stated success rate can be improved by making use of lattice-reduction techniques. Previously, we defined an attack to be successful if all n NTT inputs are correctly recovered. For the full Kyber scheme, one needs to accomplish this k times to recover all nk coefficients of $\mathbf{r}$.

One can, however, also recover the message when only using the $nk - l$ most probable coefficients, for some small l. That is, one only picks the $nk - l$ coefficients where the final probability of the most likely value is closest to 1. These values are then plugged into the equation $\mathbf{c}_1 \coloneqq \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$, the remaining l coefficients can then be recovered using lattice-reduction techniques. Such an approach was also used by Primas et al. [23], which is why we do not give further details here.

When using $l = 120$, a conversion to unique-SVP [1], and the BKZ lattice-reduction algorithm with block size 25, then the unknown coefficients can be recovered in approximately $5 - 10$ minutes. When picking $k = 3$ out of the 100 real experiments at random, then the success rate is increased to 0.95 when using this approach with the stated parameters.

7 Countermeasures

In the previous sections, we established that single-trace attacks on the NTT can be made truly practical. Several improvements to the underlying use of belief propagation in conjunction with the exploitation of small coefficients allow attacks even with simple Hamming-weight templates. This clearly shows that countermeasures are needed even for encryption where no long-term secrets are involved. We now discuss some possible options.

Masking. Masking is firstly a DPA countermeasure, but in our scenario also somewhat counteracts single-trace attacks. They are still possible, as shown in Section 5.2, but the acceptable noise level is drastically decreased. Nonetheless, further countermeasures are likely needed to protect against more sophisticated attackers.

Blinding. Saarinen [26] and Oder et al. [22] make use of a blinding technique, which can be seen as a simple form of masking. They blind the two to-be-multiplied polynomials by first multiplying them with two random scalars $(a, b) \in \mathbb{Z}_q$. The product is then unblinded via a multiplication with $(ab)^{-1} \bmod q$. Note that this scalar blinding does keep the narrow support of, e.g., $\mathbf{r}$, intact. The concrete values of the support are however changed. Such a countermeasure might not be able to prevent attacks. First, one can mount a horizontal side-channel template attack, e.g., on all n multiplications with a, to recover the fixed blinding value. Second, one can model the blinding value as an additional variable node in the factor graph and let belief propagation recover its value. We do not further study these scenarios here.

Shuffling. Similar to other algebraic side-channel attacks, shuffling is probably a very effective countermeasure. By randomizing the order of executed operations within each NTT layer, the leakage points cannot be trivially assigned to the correct variable nodes anymore. Note that shuffling linear operations, such as pointwise multiplications, was proposed by Oder et al. [22], but does not affect an attack on the NTT. We leave an analysis of the required granularity of shuffling and the overall cost of this countermeasure for future work.

Acknowledgements. This work has been supported 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, and via the project ESPRESSO, which is funded by the province of Styria and the Business Promotion Agencies of Styria and Carinthia.

References

1. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to LWE. In: ASIACRYPT (1). LNCS, vol. 10624, pp. 297–322. Springer (2017)
2. Alkim, E., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Pöppelmann, T., Schwabe, P., Stebila, D.: NewHope. Submission to [21] (2017), `https://newhopecrypto.org/`
3. Alkim, E., Bos, J.W., Ducas, L., Longa, P., Mironov, I., Naehrig, M., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: FrodoKEM. Submission to [21] (2017), `https://frodokem.org/`
4. Alkim, E., Jakubeit, P., Schwabe, P.: Newhope on ARM cortex-m. In: SPACE. LNCS, vol. 10076, pp. 332–349. Springer (2016)
5. Alperin-Sheriff, J.: Programmable hardware, microcontrollers and vector instructions. Post on the NIST pqc-forum, `https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/_OmDoyry1Ao/Tt7yHpjSDgAJ` (2018)
6. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schank, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber. Submission to [21] (2017), `https://pq-crystals.org/kyber`
7. Aysu, A., Tobah, Y., Tiwari, M., Gerstlauer, A., Orshansky, M.: Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In: HOST. pp. 81–88. IEEE Computer Society (2018)
8. Barthe, G., Belaïd, S., Espitau, T., Fouque, P., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: EUROCRYPT (2). LNCS, vol. 10821, pp. 354–384. Springer (2018)
9. Bauer, A., Jaulmes, É., Prouff, E., Wild, J.: Horizontal collision correlation attack on elliptic curves. In: Selected Areas in Cryptography. LNCS, vol. 8282, pp. 553–570. Springer (2013)
10. Bindel, N., Akleylek, S., Alkim, E., Barreto, P.S.L.M., Buchmann, J., Eaton, E., Gutoski, G., Krämer, J., Longa, P., Polat, H., Ricardini, J.E., Zanon, G.: qTESLA. Submission to [21] (2017), `https://qtesla.org`

11. Bos, J.W., Friedberger, S., Martinoli, M., Oswald, E., Stam, M.: Assessing the feasibility of single trace power analysis of frodo. In: SAC. LNCS, vol. 11349, pp. 216–234. Springer (2018)
12. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: CHES. LNCS, vol. 2523, pp. 13–28. Springer (2002)
13. de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-lwe encryption. In: DATE. pp. 339–344. ACM (2015)
14. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: CRYPTO. LNCS, vol. 1666, pp. 537–554. Springer (1999)
15. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4, `https://github.com/mupq/pqm4`
16. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptography **75**(3), 565–599 (2015)
17. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium. Submission to [21] (2017), `https://pq-crystals.org/dilithium`
18. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. LNCS, vol. 6110, pp. 1–23. Springer (2010)
19. MacKay, D.J.C.: Information theory, inference, and learning algorithms. Cambridge University Press (2003)
20. NewAE: Cw308t-stm32f, `https://wiki.newae.com/CW308T-STM32F`
21. NIST: Post-quantum cryptography standardization. `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`
22. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 142–174 (2018)
23. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: CHES. LNCS, vol. 10529, pp. 513–533. Springer (2017)
24. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively homomorphic ring-lwe masking. In: PQCrypto. LNCS, vol. 9606, pp. 233–244. Springer (2016)
25. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-lwe implementation. In: CHES. LNCS, vol. 9293, pp. 683–702. Springer (2015)
26. Saarinen, M.O.: Arithmetic coding and blinding countermeasures for lattice signatures - engineering a side-channel resistant post-quantum signature scheme with compact signatures. J. Cryptographic Engineering **8**(1), 71–84 (2018)
27. Storkey, A.J.: Generalised propagation for fast fourier transforms with partial or missing data. In: NIPS. pp. 433–440. MIT Press (2003)
28. Veyrat-Charvillon, N., Gérard, B., Standaert, F.: Soft analytical side-channel attacks. In: ASIACRYPT (1). LNCS, vol. 8873, pp. 282–296. Springer (2014)
29. Yedidia, J.S.: Sparse factor graph representations of reed-solomon and related codes. In: Algebraic Coding Theory and Information Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 68, pp. 91–98. DIMACS/AMS (2003), full version available at `http://www.merl.com/publications/docs/TR2003-135.pdf`
30. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Generalized belief propagation. In: NIPS. pp. 689–695. MIT Press (2000)