Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4

Guilhèm Assael, Philippe Elbaz-Vincent, Guillaume Reymond

Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4

Guilhèm Assael*†, Philippe Elbaz-Vincent†, Guillaume Reymond*

*STMicroelectronics Rousset, 190 avenue Celestin Coq, 13106 Rousset, France
guilhem.assael@st.com, guillaume.reymond@st.com
†Univ. Grenoble Alpes, CNRS, IF, 38000 Grenoble, France
philippe.elbaz-vincent@math.cnrs.fr

***Abstract*—The Number-Theoretic Transform (NTT) is a key feature for the efficiency of numerous lattice-based cryptographic schemes. The arithmetic structure of that operation makes it an important target for soft-analytical side-channel attacks, that are powerful single-trace side-channel attacks exploiting known arithmetic structure to improve noise tolerance. Among others, Pessl *et al.* used the belief-propagation technique to attack a software implementation of the Kyber key encapsulation mechanism for Arm Cortex-M4 microcontrollers. However, that implementation has since been thoroughly optimized, in particular through the use of an improved version of Plantard modular arithmetic. In this paper, we describe how we successfully attack the latest available version of this implementation. We show that precise knowledge of the implementation at hand allows for better performance of the belief-propagation technique. By modeling each individual arithmetic operation performed by the microcontroller, we are able to recover the secret values processed during the NTT, even with very noisy side-channel leakage. We also study some strategies for the attacker to either maximize the success rate, or minimize the runtime of the attack.**

***Index Terms*—single-trace side-channel attacks, NTT, post-quantum cryptography, CRYSTALS-Kyber, belief propagation**

I. Introduction

In response to the threat that increasingly powerful quantum computers pose to the security of current public-key cryptography, such as RSA and elliptic-curve cryptography, several organizations (e.g. [1]) have pushed for the development of new public-key cryptography algorithms that resist attacks by quantum computers: they are called Post-Quantum Cryptography (PQC). In particular, the US National Institute of Standards and Technology (NIST) started a process for the standardization of PQC schemes [2]. After three rounds of evaluating and selecting the various candidates, NIST recently announced the algorithms that will be standardized, and the ones that will be subjected to further evaluation [3]. The sole Key-Encapsulation Mechanism (KEM) that has been selected for standardization as of now, CRYSTALS-Kyber [4], is a lattice-based algorithm that is both fast and relatively compact.

However, lattice-based schemes remain vulnerable to side-channel attacks [5], [6], [7], in particular when they are combined with chosen-ciphertext attacks [8], [9]. Among these, single-trace attacks are especially troubling, since they allow an attacker to determine secret information by observing a single cryptographic operation, and are often difficult to protect against.

Notably, Primas *et al.* [10] showed a single-trace attack against the Number-Theoretic Transform (NTT), an arithmetic operation notably used in Kyber. This attack used *belief propagation*, a technique first introduced in the context of so-called soft-analytical side-channel attacks (SASCA) by Veyrat-Charvillon *et al.* [11]. Pessl *et al.* [12] improved the attack, making it more tolerant to noise and reducing the initial effort for templating the targeted implementation. Hamburg *et al.* [13] later showed that by using sparse chosen ciphertexts, noise tolerance could be further improved and some assumptions of the previous attack could be relaxed, at the cost of having to acquire several traces to recover the complete long-term secret key. Finally, Hermelink *et al.* [14] recently studied how to adapt the attack for it to handle shuffled computations, and managed to carry it out successfully in some specific cases.

Among these attacks, those that were performed on real traces targeted an ARM Cortex-M4 microcontroller, either running the reference implementation of Kyber, or running an older version of the Kyber implementation proposed by the `pqm4` project [15]. Besides being an important target for practical applications using PQC algorithms, Cortex-M4 microcontrollers have been chosen by NIST as the primary evaluation platform for embedded devices during the PQC standardization process. However, to the best of our knowledge, no such attacks target the most recent version of the Kyber implementation provided by `pqm4`, which is thoroughly optimized for Cortex-M4 microcontrollers. Hamburg *et al.* [13] hint that attacking that implementation requires some care to account for its specifics. In the following, we show that by closely modeling this implementation, we can perform very effective single-trace attacks.

Our contribution

We apply the belief-propagation attack to an optimized software implementation of the NTT making use of the specific arithmetic instructions available in the ARMv7E-M instruction set. We show that considering a reasonable

This is the author version of G. Assael, P. Elbaz-Vincent and G. Reymond, "Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4," *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA, 2023, pp. 111-121, doi: 10.1109/HOST55118.2023.10133270. https://ieeexplore.ieee.org/abstract/document/10133270

leakage model, the attack succeeds even in the presence of noise, despite the compactness of the implementation, that limits the amount of side-channel leakage. We also study the influence of measurement-noise variance on the success rate and the execution time of the attack, and show that a precise knowledge of the amount of noise is not required to get satisfactory results.

By specializing our attack to situations when the NTT is run over small-valued coefficients, we are able to recover all secret coefficients with high probability up to much stronger noise levels, while keeping the computational effort well within the capabilities of any individual attacker.

Outline

In Section II, we give the notations used in this article, and introduce the relevant background. In Section III, we detail the implementation of our attack, and evaluate its results on simulated side-channel traces in Section IV. We explore in Section V how the attack can be exploited in practice, and how to protect against it. Finally, in Section VI we conclude on our contribution and describe various possible follow-ups to our work.

II. Preliminaries

After introducing some notations and recalling the use and structure of the NTT, we briefly describe Kyber key-encapsulation mechanism, then summarize how Huang *et al.* [16] optimize the implementation of its NTT for Cortex-M4 microcontrollers, and wrap up with a description of the Belief Propagation algorithm in the context of Soft-Analytical Side-Channel Attacks.

A. Notations

We denote vectors by bold lowercase letters, e.g. $\mathbf{x}$, and matrices by bold uppercase letters $\mathbf{A}$. Given two polynomials u and v, we denote by uv their convolution product (usual polynomial product).

When $\mathcal{S}$ denotes a set, notation $u \leftarrow \mathcal{S}$ describes the sampling of quantity u uniformly at random from $\mathcal{S}$. When $\mathcal{S}$ denotes a probability distribution, that notation describes the sampling of u according to said probability distribution. When that set or that law is raised to some kth power, u is a k-element vector each of whose elements is sampled independently, or a $k \times k$ matrix when the exponent is $k \times k$.

Given a fractional number r, we define $\lceil r \rfloor$ to be the nearest integer to r, ties being rounded up. That operation is applied coefficient-wise to polynomials and polynomial vectors.

For two integers x and y expressed over some defined number of bits, $x \parallel y$ denotes the concatenation of their representative bit-strings, x being placed in the most-significant part. The top and bottom part of that concatenation are respectively written as $(x \parallel y)_{\mathsf{t}} = x$ and $(x \parallel y)_{\mathsf{b}} = y$.

For a prime q, we denote by $\mathbb{F}_q = \mathbb{Z}/q\mathbb{Z}$ the field of integers modulo q.

B. Number-Theoretic Transform

Cryptography algorithms based on module or ideal lattices make a large use of arithmetic operations on integer polynomials, in particular polynomial multiplication. To optimize the efficiency of these operations, several schemes [4], [17], [18] have parameters that allow (or mandate) the polynomial multiplications to be performed using the NTT, which is asymptotically the most efficient algorithm for that task.

The NTT is the equivalent of the Discrete Fourier Transform (DFT) for prime-order finite fields. It provides a bijective mapping from a polynomial ring R to the corresponding so-called *NTT domain*. Polynomial multiplication (or convolution) in the former domain translates to point-wise multiplication in the latter. More concretely, given two polynomials f and g from R, their product can be computed as $fg = \mathrm{NTT}^{-1}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$ where $\circ$ represents point-wise multiplication and NTT, NTT^{-1} are the NTT in the forward and inverse direction respectively.

The implementation of an n-point NTT (n being a power of two) can be done efficiently with a regular structure of so-called *butterfly* operations, arranged in several *layers*. These butterflies take as input a pair of coefficients, perform simple modular arithmetic on the pair together with one root of unity (in the sense of modular exponentiation), and output another pair of coefficients that will be processed by the next layer. The roots of unity are often called *twiddle factors*. In the case of Kyber, the NTT is processed in 7 layers, each applying 128 butterfly operations.

The NTT is often implemented with Cooley-Tukey (CT) butterflies in the forward direction, and Gentleman-Sande (GS) butterflies in the reverse direction [16]. These operations, applied to two integer values a and b and parameterized by twiddle factor ζ, are respectively described by Equations (1) and (2). We note that the GS butterfly sometimes includes modular divisions by 2, but they are often left out from the butterfly and carried out during a pre- or post-processing.

$$\mathrm{CT}_\zeta(a,b) = \big((a+\zeta b) \bmod q \quad (a-\zeta b) \bmod q\big) \tag{1}$$

$$\mathrm{GS}_\zeta(a,b) = \big((a+b) \bmod q \quad ((b-a)\zeta) \bmod q\big) \tag{2}$$

C. Kyber

Kyber [4] is a key-encapsulation mechanism based on the classical and quantum hardness of the Module Learning-With-Errors (MLWE) problem. Recently, NIST chose this KEM for standardization due to its simplicity, performance and security [3].

In all its security levels, Kyber uses matrices and vectors of polynomials from a fixed ring $R_q = \mathbb{F}_q[X]/(X^n+1)$, where the degree of the reducing polynomial is $n = 256$ and the integer modulus is prime $q = 3329$. The security level is chosen by varying the dimension k of matrices and vectors between 2 and 4. The error sampling required by the MLWE framework makes use of binomial distributions. We denote by $\mathcal{B}_\eta$ the distribution over polynomials, such that each coefficient independently follows a binomial law of parameter

η, having support $\{-\eta, -\eta+1, \ldots, \eta\}$. Since it heavily relies on polynomial arithmetic, Kyber mandates the use of the NTT in its specification, and optimizes some operations by sampling some random elements already in the NTT domain.

We now describe the simplified layout of Kyber's key generation, encryption and decryption operations[1]. We omit several aspects from this description, in particular the compression and decompression of the ciphertext, as well as the conversion from polynomial representation to byte strings and conversely.

We give in Algorithm 1 below the procedure for key generation. A private key for Kyber is given by a polynomial vector $\mathbf{s}$ whose coefficients are sampled from a binomial distribution, while the corresponding public key is a tuple containing a matrix $\mathbf{A}$ of polynomials sampled uniformly at random, and the product of the public matrix by the private vector with the addition of secret noise.

Algorithm 1: Kyber key generation (simplified)

Output: Private key $sk = \hat{\mathbf{s}} \in R_q^k$
Output: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k \times k} \times R_q^k$
$\hat{\mathbf{A}} \leftarrow R_q^{k \times k}$
$\mathbf{s} \leftarrow \mathcal{B}_{\eta_1}^k$
$\mathbf{e} \leftarrow \mathcal{B}_{\eta_1}^k$
$\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
return $sk = \hat{\mathbf{s}}$, $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}})$

Algorithm 2 below shows how the encryption of a 256-bit message m is performed: a one-time secret vector $\mathbf{r}$ is sampled from a binomial distribution, and multiplied on one hand by public matrix $\mathbf{A}$, and on the other hand by public vector $\mathbf{t}$. Before being released, these two results are added with some independently-sampled noise ($\mathbf{e_1}$ and e_2 respectively), and the second one is further added with a polynomial representation of the message (each coefficient being either 0 or $\lceil q/2 \rfloor$ depending on the corresponding message bit).

Algorithm 2: Kyber encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k \times k} \times R_q^k$
Input: Message $m \in \{0,1\}^n$
Output: Ciphertext $c = (\mathbf{u}, v) \in R_q^k \times R_q$
$\mathbf{r} \leftarrow \mathcal{B}_{\eta_1}^k$
$\mathbf{e_1} \leftarrow \mathcal{B}_{\eta_1}^k$
$e_2 \leftarrow \mathcal{B}_{\eta_2}$
$\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$
$\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^\mathsf{T} \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^\mathsf{T} \circ \hat{\mathbf{r}}) + e_2 + \lceil q/2 \rfloor m$
return $c = (\mathbf{u}, v)$

Finally, decryption (shown in Algorithm 3 below) involves multiplying the first part $\mathbf{u}$ of the ciphertext with the private vector, subtracting it from the second part v of the ciphertext, and decoding one message bit per coefficient of the resulting polynomial, by determining for each bit whether it is closer to 0 or to $\lceil q/2 \rfloor$.

[1]Kyber is specified as a KEM which is secure in the model of Indistinguishability under a Chosen-Ciphertext Attack (Ind-CCA). That KEM is built upon the underlying public-key encryption scheme composed of the key-generation, encryption and decryption algorithms we describe here.

Algorithm 3: Kyber decryption (simplified)

Input: Private key $sk = \hat{\mathbf{s}} \in R_q^k$
Input: Ciphertext $c = (\mathbf{u}, v) \in R_q^k \times R_q$
Output: Message $m \in \{0,1\}^n$
$w = v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^\mathsf{T} \circ \mathrm{NTT}(\mathbf{u}))$
$m = \lceil (2/q)w \rfloor \bmod 2$
return m

D. Optimized implementation of Kyber NTT for Cortex-M4

Being based on the ARMv7E-M 32-bit architecture, Cortex-M4 microcontrollers have a number of arithmetic instructions that can efficiently operate on half-word (16-bit) and byte (8-bit) quantities. Thanks to these, Huang *et al.* [16] propose a fast implementation of the NTT, able to perform two simultaneous butterfly operations (either CT or GS) in seven single-cycle instructions. One novelty of their implementation is the use of an improved version of Plantard modular reduction [19] with wider input range and narrower output range, thus being beneficial to lazy-reduction techniques.

We recall their solution for a CT butterfly in Algorithm 4 below. That solution is specifically tailored for Kyber's modulus $q = 3329$ using an additional constant $\alpha = 3$ such that $q2^{\alpha+1} < 2^{16}$. To be able to use that technique, the twiddle factors ζ used for the NTT need to be expressed over 32 bits and multiplied by some constant, but we do not develop that step here and refer the reader to [16] for details.

Algorithm 4: Double Cooley-Tukey butterfly for Cortex-M4

Input: Packed pairs of signed 16-bit coefficients $a = a_\mathsf{t} \parallel a_\mathsf{b}$, $b = b_\mathsf{t} \parallel b_\mathsf{b}$
Input: 32-bit corrected twiddle factor ζ (from real twiddle factor ζ_0)
Input: q and $q2^\alpha$ in two separate registers
Output: $a \equiv (a_\mathsf{t} + b_\mathsf{t}\zeta_0) \parallel (a_\mathsf{b} + b_\mathsf{b}\zeta_0)$, $b \equiv (a_\mathsf{t} - b_\mathsf{t}\zeta_0) \parallel (a_\mathsf{b} - b_\mathsf{b}\zeta_0)$
smulwb t, ζ, b // $t = \zeta b_\mathsf{b} \gg 16$
smulwt b, ζ, b // $b = \zeta b_\mathsf{t} \gg 16$
smlabb $t, t, q, q2^\alpha$ // $t = t_\mathsf{b} q + q2^\alpha$
smlabb $b, b, q, q2^\alpha$ // $b = b_\mathsf{b} q + q2^\alpha$
pkhtb $t, b, t, \mathrm{asr}\#16$ // $t = t_\mathsf{b} \parallel b_\mathsf{b}$
usub16 b, a, t // $b = (a_\mathsf{t} - t_\mathsf{t}) \parallel (a_\mathsf{b} - t_\mathsf{b})$
uadd16 a, a, t // $a = (a_\mathsf{t} + t_\mathsf{t}) \parallel (a_\mathsf{b} + t_\mathsf{b})$
return a, b

Due to its benefits, that optimized implementation is currently used for Kyber's forward and inverse NTT in the `pqm4` project [15], which proposes implementations of candidates to NIST PQC, optimized for Cortex-M4 microcontrollers.

E. Belief Propagation

We base our work upon the attack of Primas, Pessl and Mangard [10] and its improvements by Pessl and Primas [12], that use belief propagation in the context of soft-analytical side-channel attacks (SASCA). This approach starts with a side-channel template attack, that recovers probability distributions for some intermediate values manipulated by the target, depending of its leakage. Then, the operations performed by the implementation under attack are modeled, and that model is used to solve for the secret values that satisfy the leaked information.

Belief Propagation is an optimization algorithm that is well adapted to that task. We here give a high-level overview of that

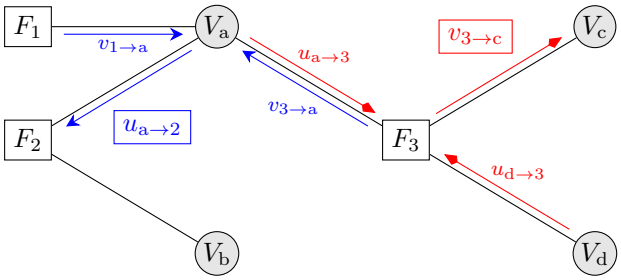

Fig. 1. Example of a factor graph with update rules depicted for two messages: from variable a to factor 2 (blue arrows) and from factor 3 to variable c (red arrows).

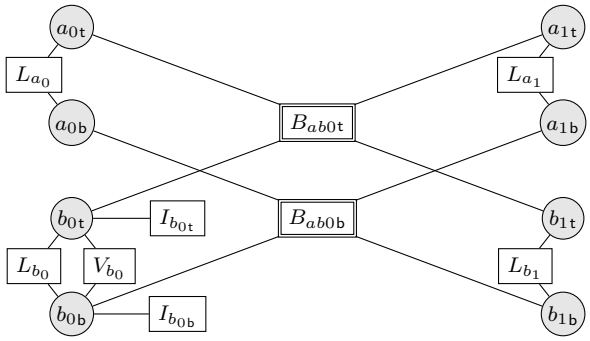

Fig. 2. Excerpt from the factor graph showing one double-butterfly operation with inputs $a_{0\mathrm{t}}||a_{0\mathrm{b}}$ and $b_{0\mathrm{t}}||b_{0\mathrm{b}}$, and outputs $a'_{1\mathrm{t}}||a'_{1\mathrm{b}}$ and $b'_{1\mathrm{t}}||b'_{1\mathrm{b}}$

technique, and refer the reader to MacKay [20] for details and definitions. The aim of belief propagation in our setup is to marginalize a joint probability distribution, that is, to derive from it independent probability distributions for each involved random variable. Practically, our joint probability distribution is the sum of the information we have on the algorithm being attacked, together with the information we get through side-channel leakage. Belief propagation then helps us determine probable values for each variable, which represents a secret or intermediate value used by the algorithm. To do so, the algorithm is modeled as an undirected graph having *variable nodes*, that represent the secret and intermediate values, and *factor nodes*, that represent the information we have on variables, in the form of joint probability distributions over subsets of them. Edges in that *factor graph* link factor nodes to the variables they depend on. The core of the belief-propagation algorithm then consists in passing *messages* between nodes, that is, local approximations of the marginal probability distributions of variables.

Let us illustrate how message passing works with a simple example. We depict in Fig. 1 a simple factor graph having three factor nodes $F_1, \ldots, F_3$ and four variable nodes $V_\mathrm{a}, \ldots, V_\mathrm{d}$. Each factor has a *potential*, that is, a function that assigns a probability to each possible tuple of outcomes for the variables it is linked to. Two types of messages are exchanged between nodes: variable-to-factor messages, where the message from variable n to factor m will be denoted by $u_{n\to m}$, and factor-to-variable messages, similarly denoted by $v_{m\to n}$ for the same pair of nodes. Both $u_{n\to m}$ and $v_{m\to n}$ represent an approximation of the probability distribution of variable n.

The rule for updating variable-to-factor messages is straightforward: the message from variable n to factor m is the point-wise product of all messages sent to variable n by factors other than m. For instance, the blue-framed message in Fig. 1 is updated though (3) for each outcome x of variable a.

$$u_{\mathrm{a}\to 2}(x) = v_{1\to\mathrm{a}}(x) v_{3\to\mathrm{a}}(x) \tag{3}$$

The update of factor-to-variable messages is more computationally expensive. The message from factor m to variable n is updated by computing the probability distribution of variable n from *(i)* the prior probability distributions of the other variables linked to factor m (as given by the message each other variable sends to factor m) and *(ii)* the joint probability distribution of all variables involved in factor m (as given by the potential of the factor). Practically, the red-framed message in Fig. 1 is updated according to (4) for each outcome x of variable c, where (x_a, x_d) in the sum runs over the Cartesian product of the possible outcomes for variables a and d.

$$v_{3\to\mathrm{c}}(x) = \sum_{x_a, x_d} F_3(x_a, x, x_d) u_{\mathrm{a}\to 3}(x_a) u_{\mathrm{d}\to 3}(x_d) \tag{4}$$

These message updates are repeatedly applied to each edge of the factor graph until convergence or another termination condition is reached[2]. Then, the marginal probability distribution of each variable is determined by point-wise multiplying all its incoming messages, and normalizing.

III. Attack Implementation

A. Factor graph

We model Kyber's optimized NTT implementation from the `pqm4` project as a factor graph, taking into account the actual operations performed by the microcontroller. In particular, the model includes that most input, intermediate and output values are processed in pairs.

We assume that each arithmetic operation leaks information on the result that it writes to a register, and that each load or store operation between RAM and registers leaks information on the corresponding data word. In that latter case, we assume a single leakage point for each pair of coefficients in each layer of the NTT.

We show in Fig. 2 our factor graph for a 1-layer forward NTT over 4 coefficients. We represent variable nodes with circles, and factor nodes with rectangles.

We use three types of factors denoted by L, V and I to encode the information acquired through side-channel leakage. These factors assign to each outcome of the corresponding variable (for type I) or pair of variables (for types L and V) the probability of the variable taking that outcome, conditioned

[2]For acyclic factor graphs, convergence to the optimal solution is guaranteed independently from the order in which messages are updated; when, instead, the graph contains cycles, the message-passing order matters and neither optimality nor convergence are guaranteed.

by the actual side-channel measurements associated with the factor. To that end, the measurements are supposed to be in the Hamming-weight metric, with Gaussian noise having a fixed standard deviation σ_F. Bayes' theorem is used to recover the desired probability from the knowledge of the measurement outcome.

- Factor type L represents one load or store operation of a pair of coefficients, and its potential is based on the measured leakage of the corresponding load or store, before or after the processing of the corresponding double-butterfly operation. That leakage corresponds to the measurement of a or b in input or output of Algorithm 4.
- Factor type V represents the operation of packing a pair of coefficients, after Plantard multiplication, into a single 32-bit register. Such factors are only included for variable pairs that are fed to the second input of a butterfly operation, and they are configured based on the leakage of the value computed at line 5 of Algorithm 4.
- Factor type I represents the leakage of the arithmetic operations that depend on a single variable; its potential is based on the measurement of the results of the two arithmetic operations that involve that variable alone; one is provided for each variable that is fed to the second input of a butterfly operation. The factor linked to even-index (resp. odd-index) coefficients is configured based on the leakage of the values computed at lines 1 and 3 (resp. 2 and 4) of Algorithm 4.

In addition to these, a fourth factor type models the butterfly operations. Following Pessl *et al.* [12], we use a single B-type factor node to model each butterfly, rather than one node for each output of each butterfly. Factors of that type assign equal probability to all tuples of two input values and two output values that respect the butterfly equation (taking into account Plantard multiplication and lazy reduction), and assign zero probability to all other tuples.

Applying that model to a 7-layer NTT over 256 coefficients gives a factor graph with 2048 variable nodes, 896 unary factors (type I), 1920 binary factors (type L or V), and 896 factors having a fanout of four (type B).

B. Message-updating order

We make use of a message-updating order that is similar to that of Pessl *et al.* [12]: we propagate messages from the first layer to the last one, then back to the first one. More specifically, we propagate messages one layer after another, where the order in each layer is the following:

1) from L, V and I factor nodes in input of the layer, to the input variable nodes;
2) then, from the input variables nodes to the B nodes of the layer;
3) then, from the B nodes to the output variable nodes;
4) then, from the output variable nodes to the L, V and I nodes linked to the output variables[3].

[3]The V and I factor nodes linked to output variables are not visible in Fig. 2 since only a single layer of the NTT is represented.

Once the messages have been passed across all layers in the forward direction, the above steps are done in reverse across all layers from the last to the first. A single back-and-forth propagation is referred to as one iteration.

In each of the steps enumerated above, all messages are independent from one another, and can be computed in any order, or even in parallel. That property allows the computation to be highly parallelized, up to 256 times, with a nearly linear performance gain.

C. Message damping

Similar to [12], we use message damping in order to get better and faster convergence. Since our factor graph contains loops, belief propagation can be unstable and make the messages oscillate. Such oscillations are detrimental to both convergence speed and accuracy, and should thus be dampened. To do so, every time a message is updated, a weighted average between its old value and the value given by the message-passing rules is used as its new value. We call *damping value* the weight δ given to the message-passing rule, while the old value of the message is given weight $1-\delta$. We empirically found a damping value of $\delta = 95\,\%$ to give satisfactory results, so we use it in all our experiments.

D. Message pruning

To improve the run-time of the algorithm, we adopt a message-pruning strategy, that only processes nonzero outcomes when computing message updates. To make full use of this technique, we also truncate low-probability outcomes to zero. Care must be taken to select a low enough threshold for this truncation to minimize the probability of suppressing the actual value of a variable from a belief. We chose a threshold of 10^{-8} times the highest probability in each message, as we empirically found that value to offer a good compromise between the success rate and the average runtime for various parameters.

We note that the efficiency of message pruning somewhat decreases when also using message damping, since the latter slows down the rate at which probabilities can decrease. However, that effect is only significant for experiments that converge in very few iterations.

E. Termination

We iterate the algorithm, repeatedly updating messages until one termination condition is met: either the update of messages changed all their values by less than a chosen threshold (set to 10^{-5} to guarantee having reached a stable state); or, one message is all-zeroes, which means the algorithm has reached an inconsistent state with no solutions; or, the number of iterations has reached a given limit (set to 100 after observing that most of the experiments terminated in less than 20 iterations). Once one of these termination conditions has been reached, the marginal probabilities of each variable are computed by multiplying all the messages coming from its adjacent factors, and normalizing.

F. Progress monitoring

During execution of the belief propagation, we keep track of the residual Shannon entropy of all variables. That measurement indicates the degrees of freedom that remain in the state after each iteration.

G. Implementation and computing resources

All parts of our simulation, including the model of the Cortex-M4 implementation and the belief-propagation algorithm, are implemented in Python (version 3.10.6). The algorithm implementing the message-update rules is carefully written to be efficient and cache-friendly, and furthermore compiled using the `numba` library [21] (version 0.55.1). The algorithm is run on an AMD EPYC 7713P processor running at 2 GHz. The CPU has 64 physical cores, but at most 32 of them were used for our algorithm (the CPU being shared with other unrelated tasks). In the following, we name *CPU time* the sum of the runtimes over all active CPU cores. Across all experiments, the memory usage peaked at 10 GB.

We do not claim our attack algorithm to be particularly efficient in runtime or memory usage, and only intend to demonstrate its practicality and how its runtime evolves with attack parameters.

IV. Results

In this section, we show that our attack allows us to recover a uniformly random polynomial in input of the NTT, even when the side-channel leakage has moderate noise, having a standard deviation of 1 (for 99 % success rate) or even 1.2 (for 90 % success rate). We recall that, since we are measuring the Hamming weight of 32-bit registers, noiseless measurements are in the range from 0 to 32, inclusively. We will discuss in Subsection V-A on what situations correspond to such amounts of noise in a practical attack.

We also show that the standard deviation of the noise does not need to be precisely known, and that slightly over- or under-estimating it during the attack does not significantly impact the quality or runtime of the secret recovery.

Finally, we specialize our setup to the case when the polynomial input to the NTT is sampled according to a binomial distribution, and are able to recover the secret perfectly in the vast majority of cases for much higher measurement noise, up to a standard deviation of 5.

A. Noise tolerance for uniformly-random input

We start by sampling coefficients uniformly at random between $-\lceil q/2 \rceil$ and $\lceil q/2 \rceil$ in input of the first NTT layer, and we simulate the execution of the NTT and measure each leakage point with some added noise, then use belief propagation on the obtained measurements to attempt to recover the input and intermediate values. We perform that procedure for various standard deviations (denoted σ_M) of the measurement noise, and sample 100 such experiments for each value of the standard deviation. For now, we assume that σ_M is exactly known, and we accordingly configure the factor nodes of types L, V and I such that $\sigma_F = \sigma_M$. These two quantities being equal, we collectively refer to them as σ. In the following, unless otherwise noted, an experiment is said to be *successful* when it gave the highest probability to the actual value of each input and intermediate variable.

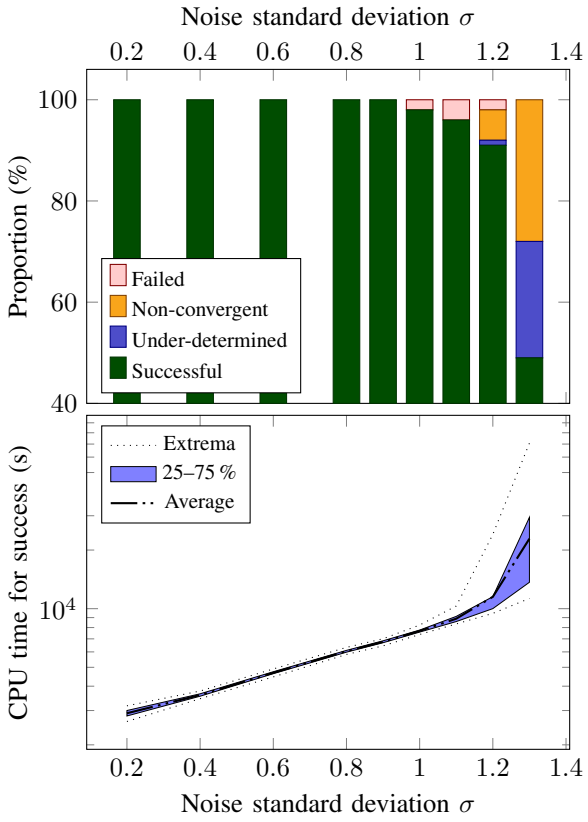

Fig. 3. Effect of noise variance on convergence rate and convergence time

The influence of the amount of measurement noise on convergence rate and convergence time can be seen in Fig. 3. For low measurement noise ($\sigma \leqslant 0.9$), all experiments are successful in less than two hours of CPU time, which only represent a few minutes of wall-clock time since we can solve the optimization problem with up to 32 CPU cores in parallel. Above that value, the success rate stays above 90 % up to a noise standard-deviation of $\sigma = 1.2$. For higher measurement noise, the success rate then quickly drops, and the runtime sharply increases, reaching an average value of six CPU hours per experiment for $\sigma = 1.3$.

Starting from $\sigma = 1.2$, the attack starts showing a variety of different behaviors: apart from *successful* experiments, in which belief propagation converges to a low-entropy state (practically, a single candidate secret has nonzero probability), and *failed* experiments, in which an inconsistent state is reached with at least one message having no positive-probability outcomes, two other behaviors are observed, and labeled as either *non-convergent* or *under-determined*. We call an experiment non-convergent when belief propagation does not terminate before the iteration limit, and we mark it as under-determined when it does terminate before that limit

but with a final state having large residual Shannon entropy (practically, between 6000 and 14 300 bits in the experiments reported in Fig. 3).

B. Influence of an incorrect estimation of the amount of noise

Since in practical attacks the standard deviation of the measurement noise is not precisely known, we study the influence of an incorrect estimation thereof on the performance of belief propagation. For a fixed actual value of the standard deviation of the noise and a fixed set of simulated traces, we try to recover their secret values using belief propagation. This time, we sample 100 executions of the NTT, with a fixed measurement noise of standard deviation $\sigma_M = 1.1$, and we attack each of them several times while varying the noise standard deviation σ_F configured for the factor nodes of belief propagation.

The results of that experiment are reproduced in Fig. 4. Let us first analyze the quality of the outcome of belief propagation. With the same termination conditions as before, we measure the quality of the attack results by the number of variables from the input layer of the NTT, whose most-probable outcome in the final state is the actual value of the variable. Since we are attacking a 256-point NTT, there are 256 first-layer variables to be recovered. As expected, the best-quality results are obtained when the noise assumed by the factor graph matches the actual measurement noise, that is, when $\sigma_F = 1.1$, represented with a dashed vertical line in Fig. 4. In that situation, all variables are successfully recovered in 99 % of the experiments, and 248 out of the 256 input variables are recovered in the remaining one.

We get the exact same quality of results when measurement noise is slightly overestimated, $\sigma_F = 1.2$. When departing from these values, the results quality progressively decreases, but more than 90 % of the experiments are still perfectly successful for $\sigma_F \in [0.8, 1.4]$. Below and above that range, the results quickly worsen, with only 164 and 144 input variables being recovered in the median case, at $\sigma_F = 0.6$ and $\sigma_F = 1.6$ respectively.

With respect to the runtime of the attack, we notice that the evolution with σ_F differs from the case when σ_F is held equal to σ_M (Fig. 3): the average runtime of the attack presents a short plateau from $\sigma_F = 0.7$ to $\sigma_F = 1.1$, reaching its minimum at $\sigma_F = 0.9$. Moreover, from $\sigma_F = 0.6$ to $\sigma_F = 1.0$, that average is strongly biased upwards due to a few experiments being non-convergent and taking much more time than usual. Consequently, the average time in that range could be lowered without significantly affecting the results quality by lowering the iteration limit to terminate non-converging experiments faster.

Combining these two parameters of runtime and result quality, we can consider two strategies available to the attacker: if it targets maximum success rate, then closely matching or slightly overestimating the measurement noise is best; if, however, the goal is to minimize the runtime while keeping exact results in the majority of cases, slightly underestimating the noise standard deviation might be best, here selecting

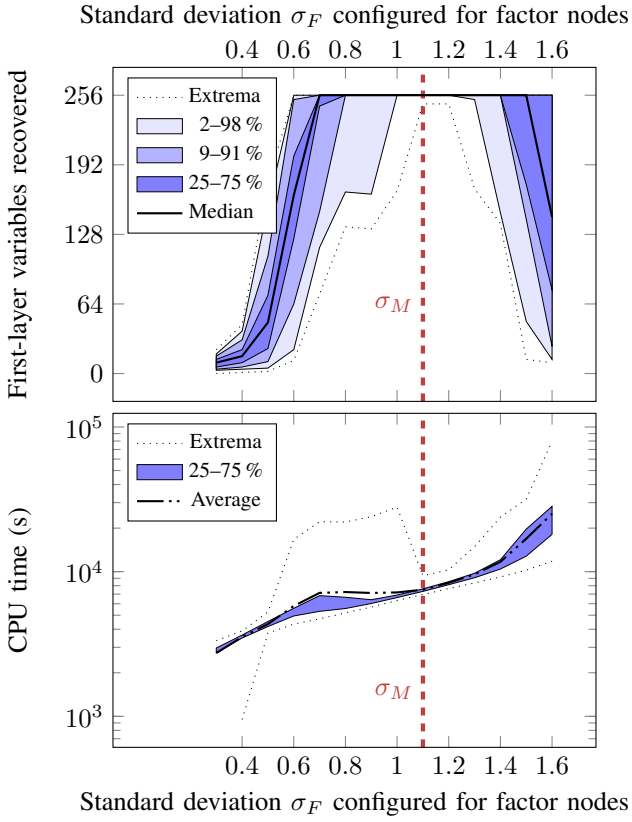

Fig. 4. Quality and runtime of the attack depending on the noise standard deviation known to the factor graph — actual noise standard deviation is $\sigma_M = 1.1$.

$\sigma_F = 0.9$, or $\sigma_F = 0.8$ with earlier termination of non-converging experiments.

C. NTT over coefficients having a small support

In the above subsections, we assumed the NTT to be run over an input polynomial having uniformly-distributed coefficients between $-\lceil q/2 \rceil$ and $\lceil q/2 \rceil$. In practice for the Kyber scheme, that situation only arises when the targeted implementation is using masked arithmetic, like the implementation of Bos *et al.* [22]. Without masking, the NTT is only ever applied to polynomials having a small support, that is, having coefficients sampled from a centered binomial distribution with small parameter. We now specialize our attack to that situation, by sampling our input polynomials from the centered binomial distribution with parameter $\eta = 3$, and adding that information into the L-type factors of the factor graph. That change of distribution is easily taken care of in the application of Bayes' theorem, with no other changes with respect to section III-A.

Since switching input coefficients from a uniform distribution to a binomial one decreases the Shannon entropy of each from 11.7 to 2.3 bits, it is expected that higher tolerance to measurement noise can be achieved. That effect is made clear in Fig. 5, where perfect accuracy is obtained for relatively large measurement noise, up to $\sigma = 4$ in all cases but one,

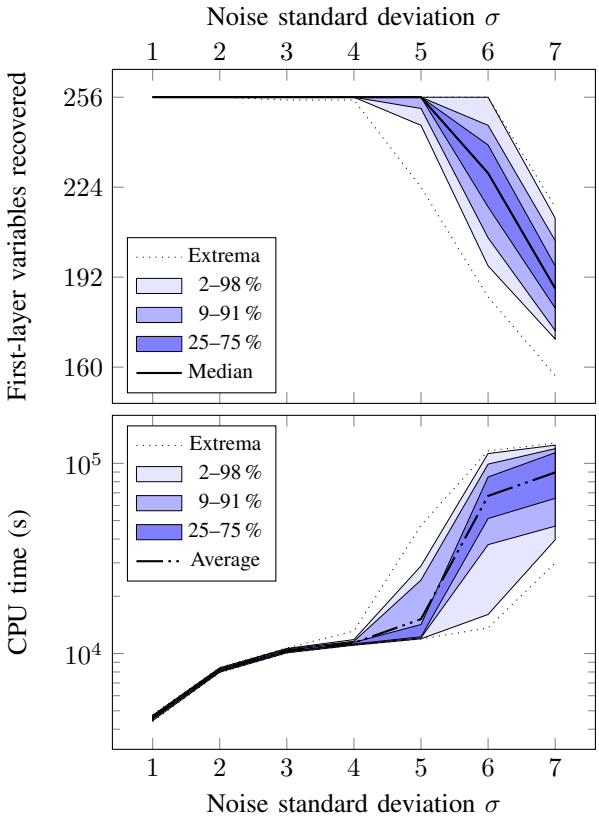

Fig. 5. Quality and runtime of the attack against an NTT over binomial coefficients with parameter 3

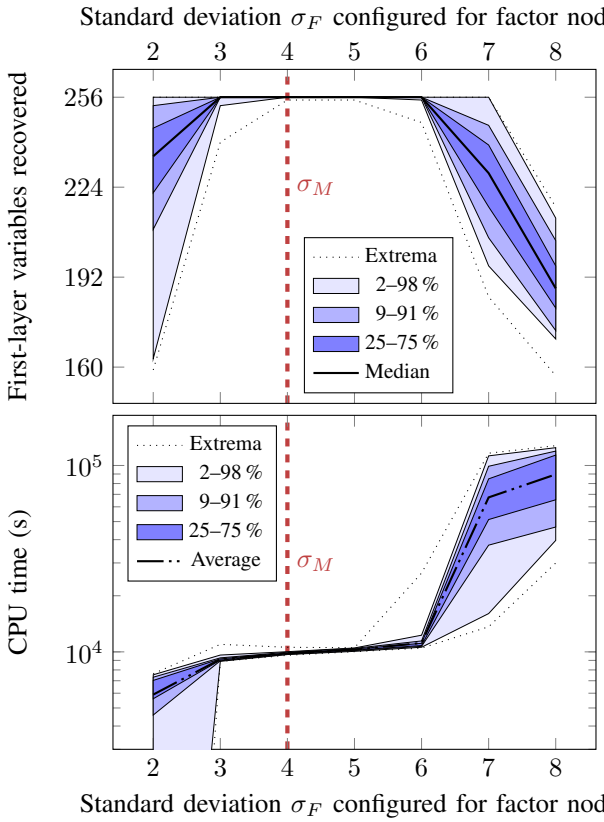

Fig. 6. Quality and runtime of the attack against an NTT over binomial coefficients with parameter 3, depending on the noise standard deviation configured in the factor graph — actual noise standard deviation is $\sigma_M = 4.0$.

and up to $\sigma = 5$ more than $75\,\%$ of the time. Above that point, the quality of the results drops: for $\sigma = 6$, only 3 out of 100 experiments recover the first layer perfectly, and 229 coefficients out of 256 are recovered in the median case. Since belief propagation often fails at recovering the first layer exactly when the noise level is that high, the runtime also increases, to an average of more than 18 CPU hours.

We note that the parameter of the binomial distribution was chosen to match the largest one used by Kyber, specifically the η_1 parameter used in its lowest security level, for sampling secret-key elements and one of the polynomial vector involved during encryption. For the other elements and for higher security levels, the binomial distribution has yet smaller support $\eta = 2$, and would allow for an even stronger attack.

Similarly to the study we did in Section IV-B, we examine how an incorrect estimation σ_F of the amount of measurement noise affects the quality and runtime of belief propagation: this time, in the case of binomially-distributed input coefficients. We set the actual standard deviation of the measurement noise to $\sigma_M = 4.0$ and run the belief-propagation algorithm with various values of σ_F from 2 to 8. The results of these experiments can be seen in Fig. 6.

In terms of results quality, we observe largely the same behavior as in the case when input coefficients are uniformly distributed. When the measurement noise is correctly configured or slightly overestimated ($\sigma_F \in [4, 5]$), $99\,\%$ of the experiments recover the first layer entirely, and the remaining one recovers 255 coefficients out of 256. In the whole range $\sigma_F \in [3, 6]$, the input coefficients are still recovered perfectly in more than $90\,\%$ of the cases. For lower and higher values of σ_F, however, the attack is less successful, with the median case recovering 235 and 229 coefficients at $\sigma_F = 2$ and $\sigma_F = 7$ respectively.

The runtime of belief propagation is closely related to the results quality, with a very regular increase in the range over which the recovery is nearly perfect: from 2.5 CPU hours at $\sigma_F = 3$ to 3.1 CPU hours at $\sigma_F = 6$. When the results quality decreases, the runtime departs from that regular tendency, falling to 1.6 CPU hour on average at $\sigma_F = 2$, and rising to 18.7 hours at $\sigma_F = 7$. Overall, we thus reiterate our remark that a close estimation of the measurement noise is not required, but deliberately over-estimating or under-estimating it can be beneficial in terms of success rate or runtime, respectively.

D. Masked implementations

To protect implementations of the NTT against multiple-trace attacks, a typical measure is arithmetic masking [23]. Using this method, each secret value is replaced with two *shares*, each of whose is chosen uniformly at random when considered independently, but whose sum is equal to the secret.

Since the NTT is a linear operation, it can be separately computed over each set of shares of the secret polynomial, and the two resulting polynomials are arithmetic shares of the expected result.

In this situation, the polynomial coefficients in input of the NTT are no longer distributed over a small support, so it seems that the attacker has to fall back to attacking the NTT over uniformly-distributed coefficients (see Subsection IV-A and Subsection IV-B). However, Pessl at al. [12] show that the information over the small support can still be used by attacking both share sets at the same time, thus using a factor graph twice the size of the previous one, with additional factor nodes connecting corresponding shares and encoding the information over the small support of their sum. Although their attack tolerates less noise in the masked case than in the unprotected case, perfect success rates can still be achieved against masked implementations having low noise.

We argue that our attack still allows for significant noise in the masked case: indeed, our setup with uniformly-distributed coefficients in input of the NTT can be applied directly to the masked case by processing shares independently, thus squaring the success rate. We are thus guaranteed to obtain nearly-perfect success up to $\sigma = 1.0$. Specializing the attack to the masked case as described by Pessl *et al.* [12] can only bring further improvements to noise tolerance.

V. Practical significance

A. Attack exploitation

Our attack can recover the polynomial provided in input to the forward-NTT operation. From Subsection II-C, we can see that the attack allows for recovering the one-time secret $\mathbf{r}$ used in encryption (thus allowing to recover the corresponding plain-text message), or the long-term private key $\mathbf{s}$ (allowing to decrypt all past and future ciphertexts for that key pair), depending on which NTT operation is targeted.

When trying to perform message recovery, the attacker should target the NTT or $\mathbf{r}$ at line 4 of Algorithm 2. Having recovered $\mathbf{r}$, it is then possible to compute the message as $m = (v - \mathbf{t}^\mathsf{T}\mathbf{r})/\lceil q/2 \rfloor$, where v is part of the released ciphertext and $\mathbf{t}$ belongs to the public key.

Much more powerful is a recovery of the long-term private key, which can clearly be carried out during key generation. Attacking the NTT of either $\mathbf{s}$ or $\mathbf{e}$ (lines 4 and 5 of Algorithm 1) leaks the private vector, either directly or through $\mathbf{s} = \mathbf{A}^{-1}(\mathbf{t} - \mathbf{e})$. This attack can thus be applied at two points during key generation; however, since key generation is only supposed to occur once for a given key pair, and since it might be performed in a more controlled environment than subsequent cryptographic operations, such realization may be difficult for the attacker.

We note that in these practical realizations, the attack must be run k times since k-element vectors have to be recovered. Consequently, for the lowest security level of Kyber ($k = 2$), the success rate of the whole attack would be the square of the success rate of attacking the NTT of each polynomial.

Since the decapsulation of Ind-CCA Kyber involves an encryption step, which is subject to the message recovery attack described above, private-key recovery can also be performed during decapsulation, with the help of chosen ciphertexts. Indeed, Ngo *et al.* [6] devised a message-recovery attack on the Saber KEM [24] and introduced techniques to convert it into a key-recovery attack when combined with a very few chosen ciphertexts. They showed 8 chosen ciphertexts to be enough when perfect message recovery is possible, or 16 when message recovery is imperfect. While their attack is applied to Saber rather than Kyber, their conversion of message recovery into private-key recovery only relies on high-level characteristics that are shared by the two schemes. We can thus expect similar efficiency for that conversion in the case of Kyber.

The noise levels that our attack can tolerate in simulation are reasonable, given that most Cortex-M4 microcontrollers have high leakage in practice. Since we are mainly dealing with register leakage, we expect that the high signal-to-noise ratio (SNR) required for the success of the attack in Subsections IV-A and IV-B can only be obtained with a high-quality measurement setup and a high templating effort, but still within the capabilities of a determined attacker. On the other hand, the low SNR that can be accommodated by the attack in Subsection IV-C can be obtained with low-cost equipment and minimal templating. We also stress out that the Hamming-weight leakage model is relevant, as the SNR in that metric generally dominates the SNR in the Hamming-distance metric on these microcontrollers.

In cases when the actual SNR is too low for the attack to succeed, the attacker may always switch to higher-quality equipment, for instance switching from power measurement to electromagnetic measurement, and increase the templating effort. In some cases, such as when attacking the decapsulation of chosen ciphertexts, it is also possible to average the acquired traces over several decapsulations of the same ciphertext, thereby allowing for arbitrarily low noise with sufficiently many traces. We can thus confidently assert that the attack presented here is bound to eventually succeed unless effective countermeasures are used against it.

B. Countermeasures

As proved in [12] and discussed above, masking is not a sufficient countermeasure to the attack since it merely decreases the maximum allowable noise. However, since the belief-propagation technique relies on correctly associating each leakage point with a particular variable (or set of variables), shuffling has already been proposed as an effective countermeasure [12]. While Hermelink *et al.* have adapted belief propagation to attacking shuffled NTTs [14], their attack assumes a much more powerful adversary, able to inject reliable zeroing faults. It is also less tolerant to noise, and most importantly, it requires either the shuffling to be partial, or the attacker to get sufficient information on the shuffling order to reduce the situation to the partial-shuffling case.

In addition to the already-high difficulty of attacking a shuffled NTT, the situation can be further worsened for the attacker by combining shuffling with masking, and selecting different shuffling orders for the two sets of shares (in the case of first-order masking). In this case, corresponding shares cannot be easily associated, so the information on the narrow support of their sum is lost. For even better protection, the shuffling can be applied to the two sets of shares simultaneously so that they are randomly interleaved, thereby forcing the attacker to consider a much larger number of possible permutations.

VI. Conclusion

In this paper, we demonstrated a side-channel attack against the NTT from a recognized software implementation of Kyber for ARM Cortex-M4 microcontrollers. By exploiting precise knowledge of the arithmetic instructions used for intermediate computations, we were able to cope with high noise levels in simulation, that should translate to effective attacks against real-life devices in a wide range of conditions.

As future work, our attack can be optimized to the case when the NTT is protected with arithmetic masking, using the technique of Pessl *et al.* [12]. By attacking both shares of the masked polynomial simultaneously and adding factors that give information on the small support of the sum of shares of each coefficient, we should be able to obtain intermediate performance between the unmasked cases of uniformly-distributed input coefficients (Fig. 3) and binomially-distributed input coefficients (Fig. 5).

Since the performance results of our attack are based on simulated traces only, a follow-up to our work will be to evaluate it in practice, with real traces. Given that we use a standard and relatively weak leakage model based on the Hamming weight of the secrets and intermediates, we expect the attack to be effective in practice. However, while we hypothesized equal noise for all leakage points in our simulation, we anticipate significant variations of the actual noise levels across different leakage points, in particular since some of them involve values stored in RAM while others involve values stored in registers. That possible unbalance will have to be taken into account for practical attacks to be best effective.

For high noise levels ($\sigma \in [4, 6]$) when input coefficients are binomially distributed, our attack correctly recovers many, but not always all of the secret coefficients. In that case, lattice reduction can be used to recover the remaining coefficients, as shown by Pessl *et al.* [12].

Finally, some room is left for optimizing the runtime and memory usage of our implementation of belief propagation, which would allow us to attack slightly more noisy instances with the same resource cost.

VII. Acknowledgments

The authors are grateful to the reviewers for their valuable comments. They would also like to thank Ruggero Susella, Patrick Haddad, Nicolas Bruneau, Bernard Kasser and Michael Miller for their helpful contributions.

References

[1] "Quantum safe cryptography and security – an introduction, benefits, enablers and challenges," White paper, European Telecommunications Standards Institute, 2015.

[2] "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," Call for proposals, National Institute of Standards and Technology, Dec. 2016. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[3] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status report on the third round of the NIST post-quantum cryptography standardization process," National Institute of Standards and Technology, 2022. [Online]. Available: https://doi.org/10.6028/NIST.IR.8413-upd1

[4] R. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber – Algorithm specifications and supporting documentation," Jan. 2021, version 3.01. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf

[5] F. Aydin, A. Aysu, M. Tiwari, A. Gerstlauer, and M. Orshansky, "Horizontal side-channel vulnerabilities of post-quantum key exchange and encapsulation protocols," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 6, Oct. 2021.

[6] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked IND-CCA secure Saber KEM implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 676–707, Aug. 2021.

[7] S. Marzougui, V. Ulitzsch, M. Tibouchi, and J.-P. Seifert, "Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all," Cryptology ePrint Archive, Report 2022/106, 2022. [Online]. Available: https://ia.cr/2022/106

[8] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 307–335, Jun. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8592

[9] B.-Y. Sim, A. Park, and D.-G. Han, "Chosen-ciphertext clustering attack on CRYSTALS-KYBER using the side-channel leakage of Barrett reduction," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21 382–21 397, 2022.

[10] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma, Eds. Cham: Springer International Publishing, 2017, pp. 513–533.

[11] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft analytical side-channel attacks," in *Advances in Cryptology – ASIACRYPT 2014*, P. Sarkar and T. Iwata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 282–296.

[12] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *Progress in Cryptology – LATINCRYPT 2019*, P. Schwabe and N. Thériault, Eds. Cham: Springer International Publishing, 2019, pp. 130–149.

[13] M. Hamburg, J. Hermelink, R. Primas, S. Samardjiska, T. Schamberger, S. Streit, E. Strieder, and C. van Vredendaal, "Chosen ciphertext k-trace attacks on masked cca2 secure kyber," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 88–113, Aug. 2021.

[14] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, "Adapting belief propagation to counter shuffling of NTTs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 1, pp. 60–88, Nov. 2022. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9947

[15] M. J. Kannwischer, R. Petri, J. Rijneveld, P. Schwabe, and K. Stoffelen. (2022, Nov.) PQM4: Post-quantum crypto library for the ARM Cortex-M4. Commit 918f379. [Online]. Available: https://github.com/mupq/pqm4

[16] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. C. Cheung, Ç. K. Koç, and D. Chen, "Improved Plantard arithmetic for lattice-based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 4, p. 614–636, Aug. 2022. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9833

[17] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange: A new hope," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, pp. 327–343.

[18] V. Lyubashevsky and G. Seiler, "NTTRU: Truly fast NTRU using NTT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 3, pp. 180–201, May 2019.

[19] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[20] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge University Press, 2003, ch. 26.

[21] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based Python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ser. LLVM '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2833157.2833162

[22] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking Kyber: First- and higher-order implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, pp. 173–214, Aug. 2021.

[23] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked ring-LWE implementation," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 683–702.

[24] A. Basso, J. M. B. Mera, J.-P. D'Anvers, A. Karmakar, S. S. Roy, M. V. Beirendonck, and F. Vercauteren, "SABER: Mod-LWR based KEM (round 3 submission)," 2021. [Online]. Available: https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf