Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption

Robert Primas, Peter Pessl, Stefan Mangard

IAIK, Graz University of Technology, Graz, Austria
`rprimas@gmail.com, {peter.pessl, stefan.mangard}@iaik.tugraz.at`

Abstract. Although lattice-based cryptography has proven to be a particularly efficient approach to post-quantum cryptography, its security against side-channel attacks is still a very open topic. There already exist some first works that use masking to achieve DPA security. However, for public-key primitives SPA attacks that use just a single trace are also highly relevant. For lattice-based cryptography this implementation-security aspect is still unexplored.
In this work, we present the first single-trace attack on lattice-based encryption. As only a single side-channel observation is needed for full key recovery, it can also be used to attack masked implementations. We use leakage coming from the Number Theoretic Transform, which is at the heart of almost all efficient lattice-based implementations. This means that our attack can be adapted to a large range of other lattice-based constructions and their respective implementations.
Our attack consists of 3 main steps. First, we perform a template matching on all modular operations in the decryption process. Second, we efficiently combine all this side-channel information using belief propagation. And third, we perform a lattice-decoding to recover the private key. We show that the attack allows full key recovery not only in a generic noisy Hamming-weight setting, but also based on real traces measured on an ARM Cortex-M4F microcontroller.
Keywords: Lattice-Based Cryptography, Side-Channel Analysis, Single-Trace Attack, Number Theoretic Transform

1 Introduction

The current public-key infrastructure is threatened by progress towards large-scale quantum computing. Constructions based on RSA, DLP, or ECC, will succumb to Shor's algorithm [32], which is able to defeat these systems in polynomial time. While estimates on the availability of powerful enough quantum computers vary greatly–they range from 15 years [18] to never [31]–the threat is still taken very seriously. This is demonstrated by, e.g., NIST's current call for post-quantum secure proposals [19] and official recommendations regarding post-quantum security from the NSA [20].

When it comes to possible post-quantum secure algorithms, lattice-based cryptography appears to be a promising option and has garnered a lot of attention over the past decade. It proved to be versatile and efficient, as there already exist practical lattice-based constructions offering basic services such as public-key encryption, digital signatures, and key exchange. Furthermore, lattices also serve as the basis for new primitives such as homomorphic encryption.

A very popular building block for lattice-based constructions is the ring-variant of the Learning with Errors problem, RLWE [16]. Recent implementations of RLWE-based public-key encryption, e.g., [7,24,29], have shown that its performance can compete with (or even exceed) that of RSA and ECC-based systems on a large set of platforms.

While these results demonstrate practicality, the implementation-security aspect of lattice-based cryptography is still a vastly unexplored and open topic. Just like any other cryptographic algorithm, an unprotected implementation of RLWE-based encryption will succumb to side-channel attacks such as Kocher's Differential Power Analysis (DPA) [14]. Due to the large number of linear operations in the en- and decryption process, masking [5] appears to be a natural fit for protecting lattice-based cryptosystems against DPA. In fact, there already exist masked implementations of lattice-based encryption [21,26,27,28]. They also show that this countermeasure can be implemented with (relatively) little resource overhead.

However, especially for public-key primitives the Simple Power Analysis (SPA) security aspect is also of high importance. This is demonstrated by, e.g., the large number of single-trace attacks targeting implementations of RSA and ECC. Yet, for lattice-based cryptography this aspect has never been analyzed thus far. As implementation techniques for RLWE-based schemes differ drastically from those of established public-key constructions, there are new potential venues for such single-trace attacks.

Our Contribution. In this work, we are first to show that single-trace attacks are indeed a threat to implementations of lattice-based cryptography. We present a new side-channel attack on lattice-based encryption that can, given sufficient leakage, recover the private key using just the side-channel observation of a single decryption. Hence, it can also be applied to masked implementations to recover each individual share, recombine them, and still perform full decryption-key recovery.

Our attack targets the computation of the Number Theoretic Transform (NTT), which is an essential building block for almost all efficient implementations of lattice-based cryptography. Thus, the attack can be ported to not only different implementations of encryption, but also to implementations of other lattice-based constructions. Furthermore, the NTT is not the first target for a DPA attack and was thus less protected in earlier works [21].

Our attack is comprised of 3 main steps. First, we perform a side-channel template matching [6] on each modular operation performed during the inverse NTT in the decryption process. In the second step, we combine the information (probabilities of intermediates) of every operation in the entire NTT. We do so by representing the FFT-like structure of the NTT as a graph and then applying the belief propagation algorithm (BP). While the use of BP in context of side-channel attacks is not new [36,12,13], it hasn't been used in the context of public-key encryption yet. In our setting, a simple implementation of BP would require an impractical amount of time. Thus, we designed several optimizations that are targeted specifically at the NTT analysis. In our third and final step, we combine the knowledge of some secret intermediate values with the public key in order to reveal the private key. Concretely, we recover the full decryption key by first reducing the size of the public key and then performing a lattice decoding.

We evaluate our single-trace key-recovery attack in two different settings. First, we determine the success rate in a generic Hamming-weight leakage model. There, our attack has a high success rate, i.e., > 0.9, with noise parameters of up to $\sigma_l = 0.4$. Second, to verify our findings in practice we use real traces from EM measurements of an ARM Cortex-M4F software implementation. In this latter scenario, we were always able to recover the decryption key. Finally, we also show that our attack performs similarly well even if masking is used.

Outline. In Section 2, we recall lattice-based encryption, efficient implementations, as well as proposed side-channel protection mechanisms. Then, in Section 3 we recall soft-analytical side-channel attacks and belief propagation as its main tool. The three steps of the attack are then described in the following sections. The first step, a side-channel analysis of the NTT, is given in Section 4. Then, in Section 5 we efficiently combine all information using belief propagation. The third and final step, i.e., lattice decoding, is given in Section 6. We present and discuss the outcome and performance of our attack in Section 7.

2 Lattice-Based Encryption and Implementation

In this section, we recall lattice-based encryption, efficient implementation techniques, and previous works on side-channel countermeasures.

2.1 Lattice-Based Public-Key Encryption

In this work, we use the RLWE-based public-key encryption scheme proposed by Lyubashevsky, Peikert, and Regev [16]. It operates with polynomials over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ and is parameterized by the tuple (n, q, σ). n denotes the dimension of the polynomials, q is the modulus for the base field $\mathbb{Z}_q$, and σ is the standard deviation for a discrete Gaussian distribution D_σ. We use boldface lowercase letters to interchangeably denote polynomials in $\mathcal{R}_q$ as well as their respective coefficient vectors. We now recall the basic encryption scheme.

Key generation. For key-pair generation, two polynomials $\mathbf{r}_1$ and $\mathbf{r}_2$ are sampled from the discrete Gaussian distribution D_σ. Next, the public key $\mathbf{p}$ is computed as $\mathbf{p} = \mathbf{r}_1 - \mathbf{a}\mathbf{r}_2$. The uniformly-random polynomial $\mathbf{a}$ is either a global domain parameter or is also included in the public key. $\mathbf{r}_2$ is the private key, $\mathbf{r}_1$ is simply discarded.

Encryption. First, the plaintext $\mathbf{m}$ is encoded as $\overline{\mathbf{m}} \in \mathcal{R}_q$. In a simple variant of encoding, the bits of $\mathbf{m}$ are simply multiplied by $q/2$. Then, three error polynomials $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \in D_\sigma$ are sampled. The ciphertext is a pair of polynomials $(\mathbf{c}_1, \mathbf{c}_2)$ with $\mathbf{c}_1 = \mathbf{a}\mathbf{e}_1 + \mathbf{e}_2$ and $\mathbf{c}_2 = \mathbf{p}\mathbf{e}_1 + \mathbf{e}_3 + \overline{\mathbf{m}}$.

Decryption. The private key $\mathbf{r}_2$ is used to compute $\mathbf{m}^\star = \mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2$. The original message $\mathbf{m}$ is then retrieved by feeding $\mathbf{m}^\star$ to a decoder. There, one computes the distance of each coefficient in $\mathbf{m}^\star$ to $q/2$. If this distance is $< q/4$, then the decoder outputs 1, otherwise 0.

The above scheme only offers CPA security [8]. Recently, Oder et al. [21] presented an extension that also offers protection against adaptive chosen-ciphertext attacks (CCA2). However, the core encryption and decryption algorithms are identical, which is why we do not further discuss their CCA2 transformation here.

2.2 Efficient Implementation

There already exists a somewhat large body of work targeting efficient implementation of the above encryption scheme. They range from FPGAs to low-resource microcontrollers and desktop CPUs (e.g., [7,11,15,23,24,29]).

In our work we use the parameter set $(n = 256, q = 7681, \sigma = 4.51)$, which was introduced by Göttert et al. [11] and is used by all of the above implementations. The concrete security level provided by this instance is still under debate and estimates vary (see, e.g., [2,10,21]). However, all our later analysis can be extended to other parameters.

Number Theoretic Transform (NTT). If q is prime, n a power of two, and $q \equiv 1 \bmod 2n$ (which is the case for virtually all previously proposed parameter sets), then there exist primitive n-th roots of unity ω_n in $\mathbb{Z}_q$. This fact allows to efficiently compute polynomial multiplication in $\mathcal{R}_q$ by means of the Number Theoretic Transform (NTT).

The NTT is essentially a Discrete Fourier Transform (DFT) in a prime field $\mathbb{Z}_q$ instead of over the complex numbers $\mathbb{C}$. Thus, this transformation is efficiently computed using the same optimizations found in, e.g., the Cooley-Tukey FFT, and runs in time $\mathcal{O}(n \log n)$. The basic building block is a butterfly, which is comprised of a modular multiplication with a certain power of the chosen primitive root, a modular addition, and a modular subtraction. A total of $n \log_2(n)/2$ butterflies are computed during the NTT, as shown in Fig. 1 with the example of a 4-coefficient NTT. The required powers of the primitive root, i.e., $\omega_n^0 \dots \omega_n^{n/2}$, are typically called *twiddle factors*. The inverse transformation (INTT) is computed by simply invoking the NTT with $\omega_n^{-1} \bmod q$. We denote $\widetilde{\mathbf{a}}$ as the NTT transformed of $\mathbf{a}$.

Multiplication of two polynomials $\mathbf{a}, \mathbf{b}$ can now be implemented as $\mathbf{c} = \mathrm{INTT}(\mathrm{NTT}(\mathbf{a}) * \mathrm{NTT}(\mathbf{b}))$, where $*$ denotes a point-wise multiplication[1]. Thus, a

[1] This explanation is slightly simplified and omits, e.g., the scaling required for the negative-wrapped convolution. For a more thorough explanation, we refer to [29].

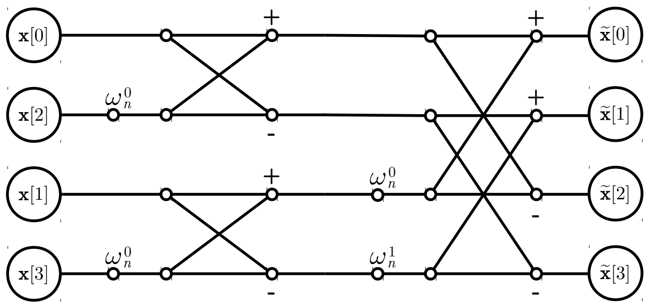


Fig. 1. A 4-coefficient NTT network comprised of 4 butterflies.

product can be computed in time complexity $\mathcal{O}(n \log n)$ (compared to $\mathcal{O}(n^2)$ for non-ring-based LWE constructions). This is one of the main arguments behind the choice of the particular ring[2] $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$.

As proposed by Roy et al. [29], the encryption scheme described in Section 2.1 can be optimized by keeping the ciphertext in the NTT domain, i.e., transmitting $(\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$. This requires that the same primitive root ω_n is used for both encryption and decryption. Thus, it must be agreed upon and is public.

2.3 Side-Channel Protection of RLWE Encryption

Implementation security of lattice-based primitives is still a very new and open topic. Yet, there already exists some previous work that studies potential protection mechanisms. We now discuss these proposals.

Masking. Due to the linearity of the main operations, i.e., polynomial addition and multiplication, the masking countermeasure [5] is a natural fit for lattice-based public-key encryption. As proposed by Reparaz et al. [27,28] and shown in Fig. 2, the private key $\mathbf{r}_2$ can be split into two shares $\mathbf{r}_2', \mathbf{r}_2''$ such that $\mathbf{r}_2 = \mathbf{r}_2' + \mathbf{r}_2'' \bmod q$. Then, polynomial multiplications, additions, and the inverse NTT can be computed on each share individually.

The final decoding step, i.e., recovering $\mathbf{m}$ from $\mathbf{m}^\star$, is nonlinear and requires more care. Reparaz et al. designed a masked decoder which outputs two binary shares of the message, i.e., $\mathbf{m} = \mathbf{m}' \oplus \mathbf{m}''$, which can then be used as a shared key in a protected implementation of, e.g., the AES.

Shuffling and blinding. In addition to masking, Oder et al. [21] propose to use further countermeasures. First, they suggest to use shuffling to protect the point-wise operations, i.e., point-wise addition and multiplication. They state that these operations are the most likely target for a DPA attack. Hence, the NTT is still computed in an unshuffled manner.

And second, they also use a randomization technique previously proposed by Saarinen [30]. They pick random values $a, b \in [1, q-1]$ and then multiply the

[2] There do exist proposals that are consciously avoiding the ring $\mathcal{R}_q$ and thus cannot use the NTT [3,4]. Still, NTT-enabled variants are the large majority.

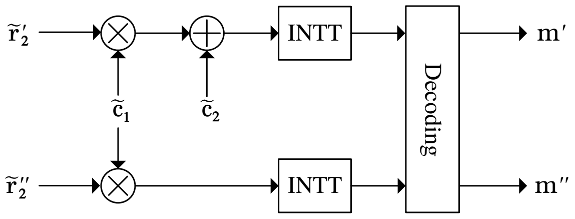

$\widetilde{\mathbf{r}}_2'$
$\widetilde{\mathbf{c}}_1$
$\widetilde{\mathbf{c}}_2$
INTT
Decoding
$\mathbf{m}'$
$\widetilde{\mathbf{r}}_2''$
INTT
$\mathbf{m}''$

Fig. 2. Basic masking scheme for decryption

coefficients $a \cdot \widetilde{\mathbf{c}}_1$, $b \cdot \widetilde{\mathbf{r}}_2$ and $ab \cdot \widetilde{\mathbf{c}}_2 \bmod q$. Due to the linearity of the NTT, the mask can be removed by multiplying the output of the INTT with $(ab)^{-1} \bmod q$.

Additively homomorphic masking. In a later work, Reparaz et al. [26] present a different masking approach which exploits the additively homomorphic property of LWE. This, however, has some caveats. First, Reparaz et al. do not claim theoretical first-order security. And second, decoding errors are more likely. This makes their method incompatible with the CCA2-transformation presented by Oder et al. [21]. Due to these reasons, we do not further analyze the susceptibility of this approach to our attack.

3 Soft-Analytical Side-Channel Attacks

In this section, we describe Soft-Analytical Side-Channel Attacks (SASCA), which were proposed by Veyrat-Charvillon et al. [36] and are one of our main attack tools. The main goal of SASCA is to bridge the gap between divide-and-conquer approaches such as Kocher's Differential Power Analysis (DPA) [14] and algebraic/analytical side-channel attacks [25]. DPA offers low time and memory complexity as well as high noise tolerance, but is suboptimal in terms of data complexity (number of observed traces). Algebraic attacks are better in this second regard, but are very sensitive to errors and often require exact information.

Veyrat-Charvillon et al. first perform a side-channel template matching [6] on all intermediates computed during an AES encryption. For each intermediate T, the template matching returns a vector of conditional probabilities $\Pr(T = t|\ell)$, where ℓ denotes the observed side-channel leakage and t runs through all realizations of the random variable T. Then, they construct a factor graph of the AES and its specific implementation. This graph models the relationship between all intermediates. Finally, they use the belief propagation algorithm (BP) on this graph and the corresponding conditional probabilities to efficiently combine the information of all leakage points. We now give a brief description of this BP algorithm.

3.1 Belief Propagation

The belief propagation algorithm, originally proposed by Pearl et al. [22], allows to efficiently compute the marginalization of a function given its factorization.

Our description and notation is largely based on that of MacKay [17, Chapter 26]. Given a function P^* of a set of N variables $\mathbf{x} \equiv \{x_n\}_{n=1}^{N}$ which is the product of M factors:

$$P^*(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m),$$

where each of the factors $f_m(\mathbf{x}_m)$ is a function of a subset $\mathbf{x}_m$ of $\mathbf{x}$ and the x_n are defined over a domain $\mathcal{D}$, the problem of marginalization is defined as computing the marginal function Z_n of any variable x_n:

$$Z_n(x_n) = \sum_{\{x_{n'}\}, n' \neq n} P^*(\mathbf{x}),$$

or its normalized marginal $P_n(x_n) = Z_n(x_n)/Z$ with the normalization constant $Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x})$. The computational cost of marginalization is believed to grow exponentially with the number of variables N. The BP algorithm aims at reducing it by exploiting the factorization of the given function. BP is based on the message-passing principle. It requires a representation of the given function as a bipartite factor graph consisting of variable nodes and factor nodes. A variable node represents one of the variables $x_i \in \mathbf{x}$, whereas a factor node corresponds to one of the factors f_{m}. Edges are drawn between x_i and f_{m} iff the factor f_{m} depends on the variable x_i. The number of variables f_{m} depends on is the factors degree $\deg(f_{\mathrm{m}})$. The BP algorithm can be used to determine the marginal functions by iteratively executing the following two steps:

From variable to factor:

$$\mathfrak{q}_{n \to m}(x_n) \;=\; \prod_{m' \in \mathcal{M}(n) \setminus \{m\}} \mathfrak{r}_{m' \to n}(x_n), \tag{1}$$

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

From factor to variable:

$$\mathfrak{r}_{m \to n}(x_n) \;=\; \sum_{x_m \setminus n} f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m) \setminus m} \mathfrak{q}_{n' \to m}(x_n), \tag{2}$$

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 as:

$$Z_n(x_n) = \prod_{m \in \mathcal{M}(n)} \mathfrak{r}_{m \to n}(x_n),$$

and the normalized marginals can be obtained from: $P_n(x_n) = Z_n(x_n)/Z$, where the normalizing constant Z is given by: $Z = \sum_{x_n} Z_n(x_n)$

If the factor graph is tree-like (acyclic), then the above algorithm returns the exact marginals. Unfortunately, in many real life applications the factor graphs contains cycles. To overcome this problem, the so called *loopy* BP algorithm has been proposed. It uses the same update rules and also iterates until convergence is reached, but uses a slightly different initialization. While it is not guaranteed that the loopy BP algorithm will return correct values or even converge, it usually gives sufficiently precise approximations of the marginals in many real-world applications. The exact conditions under which the loopy BP algorithm converges are still unknown. However, some sufficient conditions that ensure BP convergence have been stated by, e.g., Su et al. [35].

4 Attack Step 1: Side-Channels in an NTT Butterfly

After having covered all required preliminaries, we now start the description of our attack. As the first step of the attack, we exploit side-channel leakage during the computation of the inverse NTT in the decryption algorithm. Concretely, we first perform a profiling and then, for the actual attack, we match the recorded templates at each modular operation. As outcome, we obtain information in form of a probability vector for each such operation.

In order to understand how much information a side-channel adversary can realistically expect in this first step, and to also allow attack evaluation in a realistic scenario, we performed a side-channel analysis of the NTT on a real device. We now discuss our targeted implementation and platform, the measurement setup, and some results of this analysis. We additionally introduce a generic and simpler Hamming-weight leakage model, which will later be used in addition to real traces. First, however, we explain the choice of the NTT as the primary target for our attack.

4.1 The NTT as Side-Channel Target

The Number Theoretic Transform (NTT) is a main building block of virtually all efficient instantiations and implementations of lattice-based cryptography. Yet, thus far it has not been target of any side-channel analysis.

One potential reason is that the point-wise operations, i.e., multiplications and additions while computing $\widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2$, are the prime target for DPA attacks as they allow easy coefficient-wise prediction of intermediates [21]. However, this makes it tempting to use less protection in other parts, i.e., the NTT.

Also, the NTT is an interesting target for algebraic side-channel attacks. As seen in Fig. 1, it is comprised of many potentially leaking modular operations which are additionally connected by relatively simple algebraic rules. This makes it possible to combine the information of all leaking computations.

4.2 Measurement Setup and Implementation

We performed side-channel measurements on a Texas Instruments MSP432 (ARM Cortex-M4F) microcontroller on a MSP432P401R LaunchPad development board.

A Cortex-M4F was also used by many other (protected) implementations of RLWE encryption [7,21,27].

We exploit the EM side channel. As shown in Fig. 3, we placed a Langer RF-B 3-2 near-field probe in proximity to the external core-voltage regulation circuitry. This setup does not require any on-chip spatial profiling. Also, we expect similar outcomes for a power analysis. Our microcontroller was clocked at its maximum possible frequency of 48 MHz.

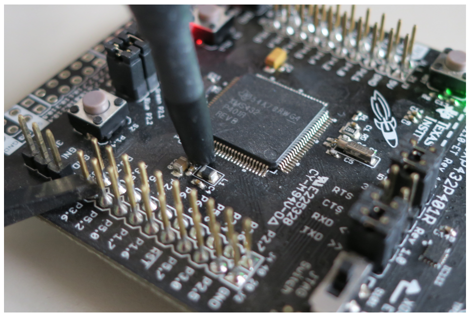


Fig. 3. EM probe placed near the voltage-regulation circuitry of an ARM Cortex-M4F

We base our analysis on the implementation techniques used in the open-sourced Cortex-M4F implementation of de Clercq et al. [7], which is also the basis of the masked software implementation of Reparaz et al. [27]. They implemented modular multiplication with division, i.e., $a \bmod q = a - q\lfloor a/q \rfloor$, and use the integrated hardware multiplier and divider. On our platform, the multiplication runs in constant time, but the `DIV` instruction does not. Reduction after addition and subtraction is implemented using ARM conditional statements (`IT` instruction), which run in constant time.

4.3 Real-Device Side-Channel Analysis

The NTT is comprised of repeated applications of a butterfly. It is a reasonable assumption that all invocations utilize the same hardware, e.g., on-chip multiplier and divider, which results in loop-invariant leakage. To simplify our later analysis and attack evaluation, we thus opt for the following approach. We analyzed the butterfly operations, i.e., modular multiplication and addition/subtraction, independently. For the analysis, the operands were preloaded into registers and no leakage of loading and storing in memory was used. We prerecorded a set number of traces for each possible operand combination. For attack evaluation, we pick a random key, perform encryption/decryption, and for each of the $n \log_2(n)/2 = 1024$ butterflies invoked during decryption randomly pick one of the prerecorded traces that corresponds to the processed intermediate. We now describe our results for each operation in the butterfly.

Modular addition and subtraction. de Clercq et al. implement modular addition and subtraction with conditional ARM statements. While these run in constant time, they still leak their state through other side-channels. With a template matching, we were able to correctly classify virtually all, i.e., > 0.99, of the taken branches. In the following, we simply assume that an attacker can correctly detect whether a reduction happened or not. Alternatively, one could also include the probability that a reduction happened in the later analysis.

Modular multiplication. In a butterfly, one of the inputs is multiplied by a known twiddle factor ω. There are $qn/2 = 983\,168$ possible operand/twiddle factor combinations, for each of them we prerecorded 100 traces. Thus, we use roughly 100 million traces for evaluation. For the attack, for each multiplication we randomly pick one out of the 100 traces corresponding to the processed value.

In the analysis, we use two steps to recover information on the unknown input. First, we exploit that the runtime of division is data dependent. We found that it depends on the bit size of the dividend, i.e., the value that is reduced (the divisor is the constant q). By measuring this time, which we do with a simple thresholding in the side-channel trace, we can immediately assign the intermediate to one out of several disjoint sets.

In the second step, we perform a side-channel template matching [6] to further narrow down the operand. For each multiplication, we use 99 (remaining) traces to build templates for each currently possible operand. The points-of-interest used for template building were determined with a t-test [9]. We then match all templates with the previously picked trace and compute the probability vector required for the next step of our attack.

In order to give a sense on the informativeness of our traces we use the metric proposed in [34], i.e., give the average entropy left in the probability vectors conditioned on the leakage $\Pr(T = t|\ell)$. Without leakage, we have an entropy of $\log_2(q) \approx 12.9$ bit. After performing the template matching, the average entropy decreases to roughly 7 bit. However, we observed that the outcome somewhat correlates with the value of the used twiddle factor. With $\omega_n^0 = 1$ we have a remaining entropy of about 10 bits. With larger values, we generally achieve better results.

4.4 A Simplified Model

In order to allow reproducibility, we additionally analyze the performance of our attack with a more generic and simpler model, namely the common noisy Hamming weight leakage model. That is, apart from knowing if a reduction happened after addition/subtraction, for each modular multiplication an attacker gets two samples of the form:

$$l = (\mathrm{HW}(a) + \mathcal{N}(0, \sigma_l))||(\mathrm{HW}(a\omega_n^i \bmod q) + \mathcal{N}(0, \sigma_l))$$

a is the unknown input and ω_n^i the used twiddle factor. HW denotes the Hamming weight function and $\mathcal{N}$ the Gaussian distribution with standard deviation σ_l. For the experiments, we perform a 2-variate template matching on these simulated traces.

5 Attack Step 2: Belief Propagation in the NTT

In the above template matching, the adversary obtains side-channel information on each computed butterfly. In the second step of the attack, we now combine all this information over the entire (I)NTT. We efficiently do so by using belief propagation. We construct a factor-graph representation of the NTT, include the side-channel information in this graph, and then run BP until convergence is reached. With the constructed factor graph the runtime of a straight-forward BP implementation is impractical. Thus, we present optimizations designed specifically for the NTT factor-graph, which decrease the runtime drastically.

5.1 Factor-Graph Construction

A factor graph is a bipartite graph containing variable nodes and factor nodes. For modeling the NTT, we add one variable node x for each input/output of a butterfly. With $n = 256$, we thus have $n(\log_2(n) + 1) = 2\,304$ variable nodes.

We then add three types of factor nodes: f_{ADD}, f_{SUB}, and f_{MUL}. As seen in Fig. 4, each type of factor occurs once per butterfly. Thus, there are a total of $3n \log_2(n)/2 = 3072$ factor nodes in the NTT model. Evidently, there are cycles in the graph shown in Fig. 4, so the loopy BP algorithm is needed.

f_{MUL} is only connected to x_2 and thus has degree 1. Its purpose is to add the side-channel information gathered from the modular multiplication of x_2 with the known twiddle factor ω. We performed a template matching in Step 1 and therefore are given vector of probabilities conditioned on the leakage l. Thus we have:

$$f_{\text{MUL}}(x_{i_2}) = \Pr(x_2 = x_{i_2}|l)$$

The factors f_{ADD} and f_{SUB} represent the modular addition and subtraction, respectively. They are connected to both butterfly-input nodes x_1 and x_2, and one of the two output nodes x_3 or x_4. Thus, their degree is 3. These factors model how variable nodes inside a butterfly are related, e.g., that $x_3 = x_1 + x_2\omega \bmod q$. Furthermore, we use these factors to include whether a reduction happened after addition or subtraction, respectively. For addition with subsequent reduction

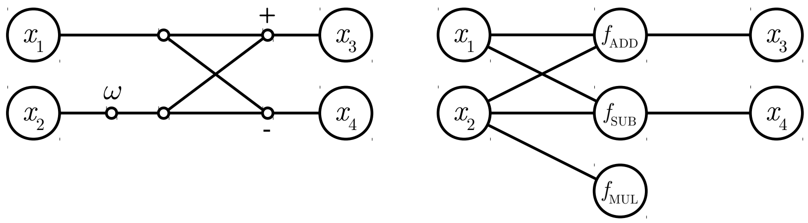


Fig. 4. Butterfly network (left) and our corresponding factor graph (right)

step, we have:

$$f_{\mathrm{ADD}}(x_{i_1}, x_{i_2}, x_{i_3}) = \begin{cases} 1 \text{ if } x_{i_1} + x_{i_2}\omega \equiv x_{i_3} \bmod q \text{ and } x_{i_1} + (x_{i_2}\omega \bmod q) \geq q \\ 0 \text{ otherwise} \end{cases}$$

If no reduction happened, then the second clause $x_{i_1} + (x_{i_2}\omega \bmod q) > q$ is simply negated. For subtraction with subsequent reduction, we have:

$$f_{\mathrm{SUB}}(x_{i_1}, x_{i_2}, x_{i_4}) = \begin{cases} 1 \text{ if } x_{i_1} - x_{i_2}\omega \equiv x_{i_4} \bmod q \text{ and } x_{i_1} - (x_{i_2}\omega \bmod q) < 0 \\ 0 \text{ otherwise} \end{cases}$$

Other leakage points. The factor-graph representation of the NTT is flexible, thus it can be modified to accommodate other leaking operations. One could, e.g., additionally include side-channel information of loading and storing in memory or leakage on operands of modular addition and subtraction.

5.2 BP Runtime Estimation without Optimization

As it turns out, the runtime of a straight-forward implementation of BP on our constructed factor graph is impractically high. It depends on the number of iterations, the number of variable nodes and the size of their domain $\mathcal{D}$, as well as the number of factor nodes and their degree.

Each iteration of BP involves the invocation of the update rules $\mathfrak{q}$ (variable to factor, Eq. 1) and $\mathfrak{r}$ (factor to variable, Eq. 2) for all variable nodes and factor nodes, respectively. In our case the number of required iterations is small, e.g., ≤ 25, and therefore does not have a significant impact on the asymptotic runtime. The runtime of $\mathfrak{q}$ is also fairly low.

However, the same cannot be said for $\mathfrak{r}$. For a factor f with degree $\deg(f)$ and its inputs $x_1, \ldots, x_{\deg(f)}$ with domain $\mathcal{D}$, one can compute the update rule given in Eq. 2 by simply looping over all $|\mathcal{D}|^{\deg(f)}$ possible input combinations of f. In our scenario, we have factors f_{ADD}, f_{SUB} with $\deg(f) = 3$ and variable nodes with domain size $|\mathcal{D}| = q = 7681$. When additionally multiplying with the number of f_{ADD} and f_{SUB} in our factor graph, then we reach a runtime of $\approx 2^{49}$ for a single iteration. Reducing from cubic to quadratic runtime can be done by only considering triplets where f_{ADD}, f_{SUB} can be 1, but this still amounts to $\approx 2^{37}$ operations. Obviously, both numbers are not very practical and optimizations are needed.

5.3 Runtime Optimizations

In Algorithm 1, we show an optimization that can decrease the runtime of $\mathfrak{r}$ for all factor nodes of degree 3 in the factor graph, i.e. f_{ADD} and f_{SUB}, drastically. We show it on the example of a factor node of type f_{ADD}. A slight variation of the presented algorithm can be used to optimize f_{SUB}.

Algorithm 1. Efficient BP for Modular Addition

```
Input:
    $\mathfrak{q}_a, \mathfrak{q}_b, \mathfrak{q}_c$   Incoming messages from summands and result node
    Reduction   True if a reduction step was executed
Output:
    $\mathfrak{r}_a, \mathfrak{r}_b, \mathfrak{r}_c$   Outgoing messages for summands and result node
1: $\tilde{\mathbf{a}} = \mathrm{FFT}_{2q}(\mathfrak{q}_a)$, $\tilde{\mathbf{b}} = \mathrm{FFT}_{2q}(\mathfrak{q}_b)$, $\tilde{\mathbf{c}} = \mathrm{FFT}_{2q}(\mathfrak{q}_c)$
2: $\mathbf{t}_a = \mathrm{IFFT}_{2q}(\mathrm{CONJ}(\tilde{\mathbf{b}}) * \tilde{\mathbf{c}})$
3: $\mathbf{t}_b = \mathrm{IFFT}_{2q}(\mathrm{CONJ}(\tilde{\mathbf{a}}) * \tilde{\mathbf{c}})$
4: $\mathbf{t}_c = \mathrm{IFFT}_{2q}(\tilde{\mathbf{a}} * \tilde{\mathbf{b}})$
5: if Reduction then
6:     $\mathfrak{r}_a = \mathbf{t}_a[q \dots 2q-1]$, $\mathfrak{r}_b = \mathbf{t}_b[q \dots 2q-1]$, $\mathfrak{r}_c = \mathbf{t}_c[q \dots 2q-1]$
7: else
8:     $\mathfrak{r}_a = \mathbf{t}_a[0 \dots q-1]$, $\mathfrak{r}_b = \mathbf{t}_b[0 \dots q-1]$, $\mathfrak{r}_c = \mathbf{t}_c[0 \dots q-1]$
```

Our optimization uses the fact that update rules for input/output distributions of modular additions/subtractions can be efficiently expressed in matrix-vector notation. Consider the addition $a + b = c \bmod q$, with $\mathfrak{q}_a, \mathfrak{q}_b, \mathfrak{q}_b$ the incoming messages from the corresponding variable nodes. Each such message is a q-dimensional vector assigning a probability to each value in $\mathcal{D}$, we say that $a_i = \mathfrak{q}_{a_i} = \Pr(a = i)$. The output $\mathfrak{r}_c$ depends on $\mathfrak{q}_a, \mathfrak{q}_b$ and an entry $c_k^* = \mathfrak{r}_{c_k}$ can be computed as the sum over all $a_i b_j$ with $i + j \equiv k \bmod q$. The whole update can be written in matrix-vector notation:

$$\begin{bmatrix} a_0 & a_{q-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & a_{q-1} & & a_2 \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ a_{q-2} & & \ddots & \ddots & a_{q-1} \\ a_{q-1} & a_{q-2} & \cdots & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{q-2} \\ b_{q-1} \end{bmatrix} = \begin{bmatrix} c_0^* \\ c_1^* \\ \vdots \\ c_{q-2}^* \\ c_{q-1}^* \end{bmatrix},$$

where the columns of the left matrix are circular shifts of $\mathfrak{q}_a$. The above equation can be rewritten as a circular convolution $\mathfrak{q}_a \star \mathfrak{q}_b$, which can be efficiently computed using the FFT and the circular convolution theorem. Thus, we have :

$$\mathfrak{r}_c = \mathfrak{q}_a \star \mathfrak{q}_b = \mathrm{IFFT}_q(\mathrm{FFT}_q(\mathfrak{q}_a) * \mathrm{FFT}_q(\mathfrak{q}_b)).$$

The update rules for $\mathfrak{r}_a$ and $\mathfrak{r}_b$ can be obtained similarly by additionally using complex conjugations CONJ, as shown in Algorithm 1. Recall that we also include whether a reduction happened during modular addition and subtraction. This can be efficiently done by replacing the q-coefficient FFT with a $2q$-coefficient FFT and by using only either the upper or lower half of the IFFT output.

The runtime of computing $\mathfrak{r}$ for the degree-3 factor nodes is now reduced to $\mathcal{O}(q \log q)$, since the only runtime relevant operations are FFTs. This allows us to perform one iteration of the BP algorithm for our whole factor graph in about one minute using a single core of an Intel Core i7-5600U CPU.

5.4 BP on Subgraphs

In our experiments, we found that applying BP to the whole NTT factor graph does not yield satisfactory results. While we can narrow down values, the outcome was not sufficient for key recovery. Yet, we were able to identify two problems and show how to circumvent them by applying BP only to subgraphs.

Uneven availability of side-channel information. The template attack on multiplication is a primary source of information. Yet, multiplications are not spread evenly across the NTT, as illustrated in Fig. 5a (also compare to Fig. 1). Each cell of this figure corresponds to one variable node. White variables are multiplied with a twiddle factor, black ones are not. Due to the lack of multiplications and its side-channel information in the top-right corner, the BP algorithm cannot recover these variable with high-enough certainty.

Varying outcome of the template attack. As already pointed out in Section 4.3, the performance of the template attack depends on the used twiddle factor. In the first NTT layer, one always multiplies with $\omega_n^0 = 1$. Even if this multiplication is not optimized out, the fact that no reduction is performed leads to little leakage.

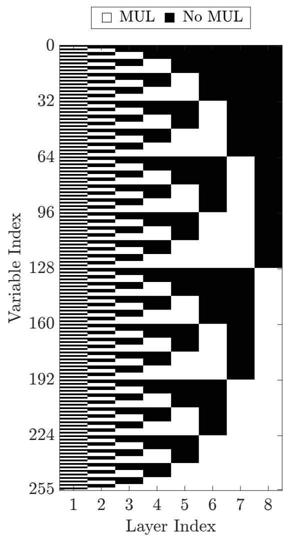
We circumvent these two problems by applying BP not on the whole NTT graph, but instead only on disjoint subgraphs. As depicted in Fig. 5b, we have subgraphs FG 1, FG 2, and FG 3. These do not include the first layer and have a higher ratio of observed to unobserved variables (compared to the full graph). Thus, applying BP to these subgraphs gives significantly better results.

After convergence is reached on all 3 graphs, we perform a classification, i.e., pick the most likely value, on certain variable nodes. Concretely, we use variables from layer 6 (output of layer 5 and input of layer 6). This is the last layer of FG 1 and variables in later layers are usually recovered with higher confidence. As shown in Fig. 5c, we use the 192 variables with indices 32...128 and 160...255.

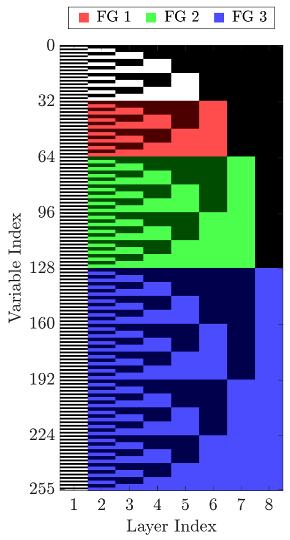
If masking is used, then we have to perform BP twice to get the intermediates in both invocations of the INTT. The unmasked intermediates can be computed by simply adding the recovered values of both INTTs.

6 Attack Step 3: Lattice Decoding

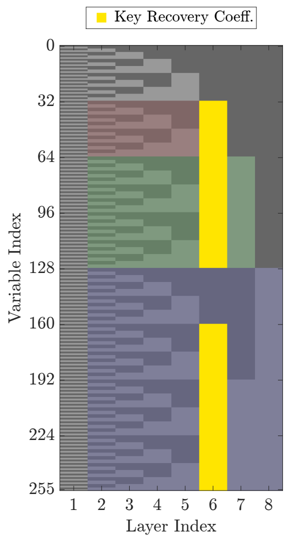
Due to applying BP only on subgraphs, we cannot recover the full INTT input $\widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2$. Hence, the decryption key $\widetilde{\mathbf{r}}_2$ (or equivalently $\mathbf{r}_2$) cannot be determined with simple linear algebra and another step is needed. In this third and final attack step, we combine the recovered intermediates with the public key. First, we create linear equations in the intermediates and $\mathbf{r}_2$ and use them to decrease the rank of the lattice spanned by the public key $(\mathbf{a}, \mathbf{p})$. Then, we use lattice-basis reduction and decoding to find $\mathbf{r}_2$ in the reduced-rank lattice.



(a) Location of multiplications



(b) Three used factor subgraphs



(c) Intermediates used for key recovery

Fig. 5. Representation of the NTT and used factors

6.1 Generating Linear Equations in the Key

We use the recovered intermediates to construct linear equations in the private key $\mathbf{r}_2$. Polynomial multiplication in $\mathcal{R}_q$ can be written as a matrix-vector product. We write the INTT output as $\mathbf{m}^\star = \mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2 = \mathbf{C}_1\mathbf{r}_2 + \mathbf{c}_2$, where the columns of matrix $\mathbf{C}_1$ are nega-cyclic rotations of $\mathbf{c}_1$. All operations inside the (I)NTT are linear, thus this system can be transformed to describe any of its intermediates. Concretely, we transform it such that it describes the recovered values of the sixth INTT layer.

We transform the system by performing a partial reversal of the INTT. We revert 3 butterfly stages by computing $x_1 = (x_3 + x_4)/2 \bmod q$ and $x_2 = (x_3 - x_4)/(2\omega)$ (cf. Fig. 4). We end up with a system of form $\mathbf{C}'_1\mathbf{r}_2 + \mathbf{c}'_2 = \mathbf{x}$, with $\mathbf{x}$ being the 192 recovered intermediates and $\mathbf{C}'_1$, $\mathbf{c}'_2$ the transformed coefficients.

6.2 Key Recovery using Lattice Reduction

The decryption key $\mathbf{r}_2$ is finally recovered by combining the above system with the information embedded in the public key $(\mathbf{a}, \mathbf{p})$. Recall that $\mathbf{p} = \mathbf{r}_1 - \mathbf{a}\mathbf{r}_2$. As $\mathbf{r}_1$ is small (it is sampled from a discrete Gaussian distribution with small σ), we have that $\mathbf{p} \approx -\mathbf{a}\mathbf{r}_2$. Thats is, $\mathbf{p}$ is close to the vector $-\mathbf{a}\mathbf{r}_2$ which is part of the q-ary lattice spanned by the columns of $\mathbf{A}$ (the matrix consisting of nega-cyclic rotations of $\mathbf{a}$). Hence, the recovery of $\mathbf{r}_2$ can be seen as a bounded-distance decoding problem. The chosen system-parameters (n, q, σ) ensure that solving this decoding problem is not feasible without further information.

However, by incorporating the linear equations from above the problem can be reduced to a size that is solvable. We substitute the 192 equations $\mathbf{C}'_1\mathbf{r}_2+\mathbf{c}'_2 = \mathbf{x}$ into $\mathbf{p} = \mathbf{r}_1 - \mathbf{A}\mathbf{r}_2$ to get some $\mathbf{p}' = \mathbf{r}_1 - \mathbf{A}'\mathbf{r}'_2$. The number of columns of $\mathbf{A}'$, and hence the rank of the spanned lattice, is now reduced to $256 - 192 = 64$.

We then search for the closest vector to $\mathbf{p}'$ by solving a shortest-vector problem. Concretely, we search for the error term $\mathbf{r}_1$ (or $-\mathbf{r}_1$) as an unusually short vector in the lattice generated by $(\mathbf{A}'||\mathbf{p}')$. This approach of solving the lattice decoding problem is described by, e.g., Albrecht et al. [1]. The short vector is recovered using the BKZ lattice basis reduction algorithm, we use the implementation provided by Shoup's NTL [33]. We invoke BKZ with a blocksize of 25, but abort reduction as soon as a candidate for $\mathbf{r}_1$, i.e., a vector with a small enough norm, is found.

After that, one can compute the private key $\mathbf{r}_2$ by solving the linear system $\mathbf{p} = \mathbf{r}_1 - \mathbf{a}\mathbf{r}_2$ for both recovered $\mathbf{r}_1$ and $-\mathbf{r}_1$. The correct $\mathbf{r}_2$ is the one that follows the distribution used for key generation. That is, we pick the smaller out of the two solutions.

Performance of decoding. We tested the correctness and performance of this key-recovery approach by performing well over 1 000 experiments. In each of them we use the correct intermediates (cf. Fig. 5c) and only perform the decoding step. All our experiments were successful. The average runtime on a single core of a Xeon E5-2699v4 CPU is approximately 45 seconds.

This decoding approach is not limited to using exactly 192 recovered intermediates, it can be invoked with any number of coefficients. However, the runtime of decoding will increase if fewer values are available. For instance, with 160 recovered intermediates the average runtime is 5 minutes and thus still well within practicality. Below that, however, it increases drastically. With 150 values, it reaches multiple hours. Experiments with 146 or fewer coefficients were not successful after 1 full day of computation.

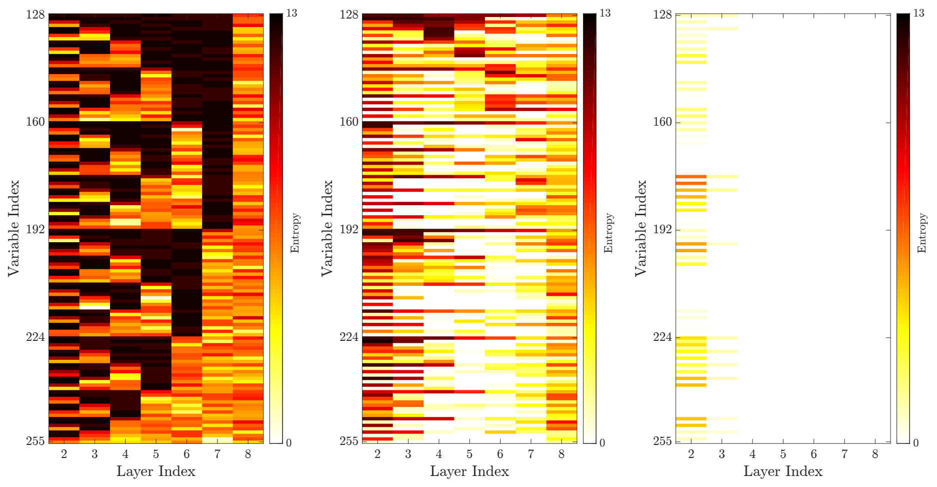
7 Attack Results and Conclusion

Our attack consists of subsequent execution of the three attack steps described in the previous sections. We now present the outcome. First, we evaluate the attack using real traces. We illustrate an exemplary outcome and give a success rate. Then, we give the success rate for the Hamming-weight model with varying noise-parameter σ_l, both with and without masking applied.

7.1 Real Device

With real traces obtained from the setup described in Section 4, we have the following results. Fig. 6 illustrates an exemplary outcome of template-matching and the subsequent belief propagation on the subgraph FG 3 (cf. Section 5.4). For each variable node, we color-code the entropy of the probability vector. For black nodes, the probability distribution is close to uniform, whereas for white nodes one value has reached probability close to 1. After 1 iteration (Fig. 6a), the probability distributions essentially correspond to the direct output of the template matching. After 20 iterations of BP (Fig. 6c), the network has converged and almost all intermediates are determined with very high probability.

Lattice decoding is successful if all of the 192 intermediates used for key-recovery are correct. After observing Fig. 6, it should not come as a surprise that all our key-recovery experiments in the real-trace setting were successful. The success rate, i.e., the probability that all used coefficients are correct, is 1.



(a) 1 iteration (b) 4 iterations (c) 20 iterations

Fig. 6. FG 3: entropy after set number of iterations of BP

7.2 Hamming-Weight Model

In order to get a broader and more generic analysis of our attack, we also tested it with a noisy Hamming-weight model (cf. Section 4.4). We rerun all tests with varying noise parameter σ_l. The outcome is illustrated in Fig. 7, where we show the success rate and the average entropy (after template matching) for each tested value of σ_l. We give the entropy to allow at least a rough comparison to the real-trace setting.

In the non-masked case, we have a high single-trace success rate up to $\sigma_l = 0.4$ or 0.5, then it drops drastically. Note, however, that an attacker that can observe multiple decryptions can decrease the observed σ_l by averaging the traces. In the masked setting, key-recovery is successful if the correct intermediates are recovered in both invocations of the inverse NTT (see Fig. 2). Only then their sum is equal to the unmasked value. Thus, the expected success rate is squared, which is confirmed by our results. Obviously, averaging cannot be done if masking is used.

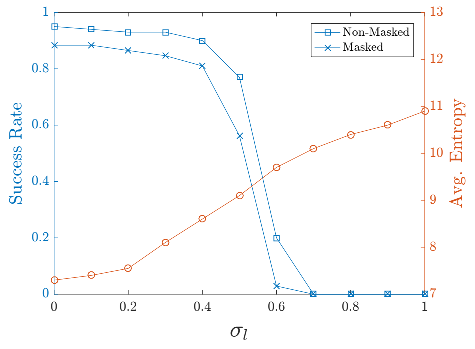


Fig. 7. Success rates in the Hamming-weight leakage model

7.3 Conclusion

Our attack clearly shows that SPA security of lattice-based schemes cannot be neglected and that relying on masking alone is not sufficient. Implementation techniques that are vastly different to established constructions such as RSA and ECC open up new venues in this regard. In fact, the regular structure of the NTT allows to efficiently combine leakage of the entire decryption process. Furthermore, each recovered intermediate can be used to decrease the difficulty of key-recovery with the public key. And while this work focuses on lattice-based encryption, our attack can be adapted to any other implementation of lattice-based cryptography which employs the NTT.

When it comes to potential countermeasures, masking appears to be effective against DPA, yet it does not prevent our attack. Thus, additional countermeasures should be implemented and will now be discussed.

Possible countermeasures. One of the first measures to strengthen an implementation against SPA attacks is to ensure a constant runtime and control flow. In our side-channel analysis of a real device, we exploit timing differences stemming from the `DIV` operation invoked during modular reduction. There do exist constant-time alternatives, as already shown by Oder et al. [21].

Like many other algebraic attacks, our key recovery can be thwarted by employing shuffling. Concretely, the operations inside the NTT, e.g., the order in which the butterflies are processed within one NTT layer, need to be shuffled. Shuffling only point-wise operations, as proposed by Oder et al., clearly does not hamper our attack. Other hiding countermeasures, such as the random insertion of dummy operations inside the NTT, can also make our attack harder.

Oder et al. also propose to use a blinding countermeasure (cf. Section 2.3). Our attack still applies, but needs an additional step and potentially a different selection of recovered intermediates. Concretely, it requires that a sufficient amount of the INTT output coefficients are recoverable or can be computed from the recovered intermediates. Then, one can test if the distribution of the unblinded INTT output, i.e., after multiplication with $ab^{-1} \bmod q$, corresponds to that of a valid $\mathbf{m}^\star$ (centered around 0 and $q/2$). For a non-masked implementation, or if the same blinding values a, b are reused for both shares, then one can run through all $q-1$ possibilities of $ab \bmod q$. If different a, b are used for both shares, then one needs to try all $(q-1)^2$ combinations. With our parameters, this can be easily done within a minute. When using 64 output coefficients, this always returned the correct blinding values in our tests. Hence, this countermeasure does not significantly increase single-trace security. It, however, prevents averaging in the non-masked scenario.

Acknowledgements
This work has been supported by the Austrian Research Promotion Agency (FFG) under project SCALAS (grant number 845589) and under the COMET K-Project DeSSnet (grant number 862235).

References

1. M. R. Albrecht, R. Fitzpatrick, and F. Göpfert. On the efficacy of solving LWE by reduction to unique-svp. In H. Lee and D. Han, editors, *ICISC 2013*, volume 8565 of *LNCS*, pages 293–310. Springer, 2013.
2. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
3. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. NTRU prime. *IACR Cryptology ePrint Archive*, 2016:461, 2016.
4. J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *CCS 2016*, pages 1006–1018. ACM, 2016.
5. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
6. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. J. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.
7. R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-lwe encryption. In W. Nebel and D. Atienza, editors, *DATE 2015*, pages 339–344. ACM, 2015.
8. S. R. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016.
9. B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. stochastic methods. In L. Goubin and M. Matsui, editors, *CHES 2006, 8th International*, volume 4249 of *LNCS*, pages 15–29. Springer, 2006.

10. F. Göpfert, C. van Vredendaal, and T. Wunderer. A quantum attack on lwe with arbitrary error distribution. Cryptology ePrint Archive, Report 2017/221, 2017.
11. N. Göttert, T. Feller, M. Schneider, J. A. Buchmann, and S. A. Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In E. Prouff and P. Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 512–529. Springer, 2012.
12. V. Grosso and F. Standaert. ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 291–312. Springer, 2015.
13. V. Grosso and F. Standaert. Masking proofs are tight (and how to exploit it in security evaluations). *IACR Cryptology ePrint Archive*, 2017:116, 2017.
14. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
15. Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede. Efficient ring-lwe encryption on 8-bit AVR processors. In T. Güneysu and H. Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 663–682. Springer, 2015.
16. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010.
17. D. J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.
18. M. Mariantoni. Building a superconducting quantum computer. Invited Talk at PQCrypto 2014, October 2014. https://www.youtube.com/watch?v=wWHAs--HA1c.
19. NIST. Post-Quantum crypto standardization, December 2016. http://csrc.nist.gov/groups/ST/post-quantum-crypto/call-for-proposals-2016.html.
20. NSA/IAD. CNSA Suite and Quantum Computing FAQ, January 2016. https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm.
21. T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Cryptology ePrint Archive*, 2016:1109, 2016.
22. J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, pages 133–136. AAAI Press, 1982.
23. T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In T. Lange, K. E. Lauter, and P. Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 68–85. Springer, 2013.
24. T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, 2015.
25. M. Renauld and F. Standaert. Algebraic side-channel attacks. In F. Bao, M. Yung, D. Lin, and J. Jing, editors, *Inscrypt 2009*, volume 6151 of *LNCS*, pages 393–410. Springer, 2009.
26. O. Reparaz, R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Additively homomorphic ring-lwe masking. In T. Takagi, editor, *PQCrypto 2016*, volume 9606 of *LNCS*, pages 233–244. Springer, 2016.
27. O. Reparaz, S. S. Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede. Masking ring-lwe. *J. Cryptographic Engineering*, 6(2):139–153, 2016. Extended journal version of [28].

28. O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede. A masked ring-lwe implementation. In T. Güneysu and H. Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 683–702. Springer, 2015.
29. S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact ring-lwe cryptoprocessor. In L. Batina and M. Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 371–391. Springer, 2014.
30. M.-J. O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, pages 1–14, 2017.
31. A. Shamir. Financial cryptography: Past, present, and future. Invited Talk at Financial Cryptography 2016, February 2016. Summary available at https://www.lightbluetouchpaper.org/2016/02/22/financial-cryptography-2016/#comment-1456744.
32. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.
33. V. Shoup. NTL: A library for doing number theory. http://www.shoup.net/ntl/.
34. F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.
35. Q. Su and Y. C. Wu. On convergence conditions of gaussian belief propagation. *IEEE Transactions on Signal Processing*, 63(5):1144–1155, March 2015.
36. N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Soft analytical side-channel attacks. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 282–296. Springer, 2014.