Chapter 2
Secure Implementation of Lattice-Based Encryption Schemes

Tobias Oder, Tobias Schneider, and Tim Güneysu

2.1 Introduction

Cryptographic key-exchange mechanisms (KEMs) are essential to secure confidential information that gets transmitted over an insecure channel. By using a KEM, the communicating parties can agree on a shared secret key that they can use to encrypt data without an eavesdropper being able to derive any information about that key. The majority of KEMs that are in use today base their cryptographic security on either the prime factorization problem or the discrete logarithm problem. However, given a fairly powerful quantum computer, one can break cryptographic schemes based on these mathematical problems using Shor's algorithm [52]. The ubiquitous threat posed by recent advances in quantum computing to currently employed KEMs

The majority of the work was performed while Tobias Schneider was with Ruhr-Universität Bochum.

T. Oder
Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
e-mail: tobias.oder@rub.de

T. Schneider
ICTEAM/ELEN/Crypto Group, Université Catholique de Louvain, Louvain, Belgium
e-mail: tobias.schneider@uclouvain.be

T. Güneysu (✉)
Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany

DFKI, Bremen, Germany
e-mail: tim.gueneysu@rub.de

R. Drechsler, M. Soeken (eds.), *Advanced Boolean Techniques*,
https://doi.org/10.1007/978-3-030-20323-8_2

has therefore caused a massive rise in research activities in the area of *post-quantum cryptography* (PQC) in the last couple of years. These research efforts culminated in the NIST requesting candidates for quantum-secure cryptographic algorithms in December 2016 [36] and releasing the list of submissions in December 2017.

The KEMs submitted to NIST base their security on entirely different security assumptions that (by the current state of knowledge) cannot be attacked by quantum algorithms. The majority of submitted KEMs is either based on linear codes, multivariate quadratics, or lattices. Lattice-based schemes constitute the largest group with 20 submitted KEMs in the first round of the standardization process and still nine schemes in the second round. There are a couple of reasons for the popularity of lattice-based cryptosystems. They offer reasonable parameter sizes, high efficiency, and flexibility regarding their potential applications. The most commonly used mathematical problem to build lattice-based KEMs is the learning with errors (LWE) problem or its variant ring-LWE that works with polynomial rings over finite fields and leads to even more efficient parameters.

With the advent of a Smart World and the Internet of Things, billions of devices are becoming connected, exchanging massive amounts of data, often combined with high demand on data security. Embedded applications of cryptography are therefore of major relevance for today's world. At the same time, these embedded targets that require the application of security solutions often have access to only limited resources. Implementations on embedded and constrained devices therefore have the requirement to be efficient in terms of performance and memory. Furthermore, in many embedded use cases an attacker has physical access to a device that contains a secret key. This physical access can then be used to extract information about intermediate values that appear during the execution of a cryptographic operation by observing physical properties of the device. For instance, the dynamic power consumption of the device can leak sensitive information about the secret key that is used in the cryptographic operation. By collecting a large number of power traces and analyzing them by statistical means, an attacker might be able to break a KEM even if it is mathematically sound. Implementations targeting embedded devices therefore also have to take into account side-channel attacks and apply appropriate countermeasures.

Because of the high efficiency, schemes based on ring-LWE can be easily implemented on embedded devices as shown by many publications, like [1, 27, 29, 30, 50]. Most of the aforementioned implementations are only protected against timing side-channels, i.e., the execution time of the implementation is independent from any secret data. In this chapter, we show how to further on protect KEMs based on ring-LWE against power analysis. To do so, we utilize masking, a common countermeasures against power attacks. Masking schemes for ring-LWE KEMs have already been investigated by Reparaz, Roy, Vercauteren, and Verbauwhede in [43, 46] and Reparaz, de Clercq, Roy, Vercauteren, and Verbauwhede in [45]. Our solution significantly improves the proposed masking schemes in terms of performance and failure rate and is also provably secure. To show the practicality of our approach, we also present an implementation on an ARM Cortex-M4 and conduct practical measurements that support our claims.

2.2 Background

2.2.1 Notation

Unless explicitly stated, we denote addition (resp. subtraction) modulo q with $+$ (resp. $-$). We denote multiplication by $\cdot$ and point-wise multiplication by $\circ$. We use $\oplus$ as operator for addition modulo 2. Polynomials in $\mathcal{R}_q = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ are labeled by bold lowercase letters. Polynomials in NTT domain are additionally marked with a tilde ($\tilde{\mathbf{a}}$). When we access a single bit of a bit vector, we use an index in square brackets to identify the respective bit.

2.2.2 Cryptographic Background

In the following, we introduce the necessary cryptographic concepts.

Key Encapsulation Mechanisms Encrypted communication can either be conducted using symmetric or asymmetric cryptography. In symmetric cryptography both parties share the same secret key that they use to encrypt their communication with. In asymmetric cryptography on the other hand, each party has a pair of keys consisting of one private key that is only known to the owner of the key and one public key that is also known to potential communication partners. Symmetric cryptography is much more efficient and therefore the primary choice to realize encrypted communication. The major drawback however is that both parties have to agree on a secret key in such a way that an eavesdropper does not get any information about that key. Therefore asymmetric cryptography is usually used to agree on a shared, symmetric key and all further communication is then encrypted symmetrically. The asymmetric primitives used to exchange a symmetric key are called Key Encapsulation Mechanisms (KEMs).

Hash Functions A cryptographic hash function is a one-way function that maps data of arbitrary size to data of fixed size. The most important properties of cryptographic hash functions are pre-image resistance, second pre-image resistance, and collision resistance. Pre-image resistance describes the one-way property of a hash functions, i.e., the difficulty to find an input that if hashed matches a given output. Second pre-image resistance on the other hand means that given a certain input, it is difficult to find another (different) input that generates the same hash value. Collision resistance means that it is hard to find *any* pair of different inputs that generate the same hash value. If the output is not fixed but variable, the algorithm is called an extendable-output function (XOF). Hash functions are also often used to instantiate random oracles in cryptographic schemes.

Security Model The security of encryption schemes (or KEMs) can be analyzed regarding different attacker models. The simplest attacker model is to assume that the attacker has access to some plaintext–ciphertext pair and tries to deduce

information about the secret key from the given pair. The difference to a chosen-plaintext attack (CPA) setting is that in this case the attacker has access to a plaintext–ciphertext pair that was generated from a plaintext that the attacker was able to choose. An even stronger assumption is the chosen-ciphertext (CCA) setting. In this case, the attacker has access to a decryption oracle and can generate plaintext–ciphertext pairs by performing a decryption operation on a ciphertext of his choice. In an *adaptive* chosen-ciphertext attack (CCA2), the attacker can even adapt his queries to the decryption oracle.

Number-Theoretic Transform The number-theoretic transform (NTT) is a discrete Fourier transform over a finite field. An interesting property of the discrete Fourier transform, which is also highly interesting for lattice-based cryptography, is the ability to reduce the overall complexity of (polynomial) multiplication to $\mathcal{O}(n \cdot \log n)$. To allow efficient computation of the NTT the coefficient ring has to contain primitive roots of unity.

Definition 2.1 (Primitive Roof of Unity [57]) Let $\mathcal{R}$ be a ring, $n \in \mathbb{N}_{\geq 1}$, and $\omega \in \mathcal{R}$. The value ω is an n-th root of unity if $\omega^n = 1$. The value ω is a primitive n-th root of unity (or root of unity of order n) if it is an n-th root of unity, $n \in \mathcal{R}$ is a unit in $\mathcal{R}$, and $\omega^{n/t} - 1$ is not a zero divisor for any prime divisor t of n.

For a given primitive n-th root of unity ω in $\mathbb{Z}_q$, the NTT of a vector $\mathbf{a} = (a_{n-1}, \ldots, a_0)$ is the vector $\tilde{\mathbf{a}} = (\tilde{a}_{n-1}, \ldots, \tilde{a}_0)$ that is computed as

$$\tilde{a}_i = \textstyle\sum_{0 \leq j < n} a_j \omega^{ij} \bmod q, i = 0, 1, \ldots, n-1.$$

The idea is to transform two polynomials $a = a_{n-1} \cdot x^{n-1} + \cdots + a_0$ and $b = b_{n-1} \cdot x^{n-1} + \cdots + b_0$ into their NTT representations $\tilde{\mathbf{a}} = \tilde{a}_{n-1} \cdot x^{n-1} + \cdots + \tilde{a}_0$ and $\tilde{\mathbf{b}} = \tilde{b}_{n-1} \cdot x^{n-1} + \cdots + \tilde{b}_0$ and computing the coefficient-wise multiplication as $\tilde{\mathbf{c}} = \sum_{0 \leq i < n} \tilde{a}_i \cdot \tilde{b}_i \cdot x^i$. The result $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ is obtained after applying the inverse transform to $\tilde{\mathbf{c}}$. For $q = 1 \bmod 2n$ the way the result has to be interpreted depends on the input.

- Assuming one expanded $\mathbf{a}$ and $\mathbf{b}$ to vectors of length $2n$ by padding n zeros, the result $\mathbf{c}$ equals the schoolbook multiplication of $\mathbf{a}$ and $\mathbf{b}$ without reduction.
- Without padding, the result $\mathbf{c}$ is already reduced modulo $f = x^n - 1$. This is called the *positive* wrapped convolution. In contrast to the first case, the resulting polynomial is only of degree n.

This reduction for free is beneficial concerning the computation time, but in schemes based on ring-LWE arithmetic is performed in $\mathbb{Z}[x]/\langle x^n+1\rangle$. Thus, the input and output have to be modified so that the *negative* wrapped convolution gets computed to exploit the reduction property. Let ψ be the square root of ω. Now one computes $\mathbf{a}' = \sum_{0 \leq i < n} a_i \cdot \psi^i \cdot x^i$ and $\mathbf{b}' = \sum_{0 \leq i < n} b_i \cdot \psi^i \cdot x^i$ before the polynomials are transformed into their NTT representation. To obtain $\mathbf{c} = \mathbf{a} \cdot \mathbf{b} \bmod x^n + 1$, one also has to multiply $\mathbf{c}'$, the output of the inverse transform (INTT) of $\tilde{\mathbf{c}}$, by powers of the *inverse* of ψ.

Learning with Errors In 2005, Oded Regev introduced the learning with errors problem [42]. The idea behind the LWE problem is to find a secret vector $\mathbf{s}$ given a public matrix $\mathbf{A}$ and a vector $\mathbf{b} = \mathbf{As} + \mathbf{e}$, where $\mathbf{e}$ is a noise vector with small coefficients. In [33], Lyubashevsky et al. present a ring variant of the LWE problem over finite fields, called ring-LWE. The (ring-)LWE problem can be used to create encryption schemes as shown in [32, 34, 35]. Several variants of the scheme exist and the concrete instantiation we are using is defined as follows:

- RLWE.CPA $^{\text{NTT}}_{\text{gen}}()$: Sample the binomial noise $\tilde{\mathbf{r}}_1 \xleftarrow{\$}$NTT(SampleNoisePoly()), $\tilde{\mathbf{r}}_2 \xleftarrow{\$}$NTT(SampleNoisePoly()), sample uniform $\tilde{\mathbf{a}} \xleftarrow{\$}$ SampleUniformPoly(), and compute $\tilde{\mathbf{p}} = \tilde{\mathbf{r}}_1 - \tilde{\mathbf{a}} \circ \tilde{\mathbf{r}}_2$. Output the secret key $\tilde{\mathbf{r}}_2$ and the public key $(\tilde{\mathbf{p}}, \tilde{\mathbf{a}})$.
- RLWE.CPA $^{\text{NTT}}_{\text{enc}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, m_{cpa} \in \{0,1\}^n)$: Sample $\tilde{\mathbf{e}}_1 =$NTT(SampleNoisePoly()), $\tilde{\mathbf{e}}_2 =$ NTT(SampleNoisePoly()), and $\tilde{\mathbf{c}}_1 = \tilde{\mathbf{a}} \circ \tilde{\mathbf{e}}_1 + \tilde{\mathbf{e}}_2$ and compute $\tilde{\mathbf{h}}_2 = \tilde{\mathbf{p}} \circ \tilde{\mathbf{e}}_1$, $\mathbf{e}_3 \leftarrow$ SampleNoisePoly(), and $\mathbf{c}_2 =$ INTT$(\tilde{\mathbf{h}}_2) + \mathbf{e}_3 +$LWEEncode$(m_{cpa})$. Output the ciphertext $(\tilde{\mathbf{c}}_1, \mathbf{c}_2)$.
- RLWE.CPA $^{\text{NTT}}_{\text{dec}}(\tilde{\mathbf{r}}_2, \tilde{\mathbf{c}}_1, \mathbf{c}_2)$: Output LWEDecode(INTT$(\tilde{\mathbf{c}}_1 \circ \tilde{\mathbf{r}}_2) + \mathbf{c}_2) \in \{0,1\}^n$.

In the scheme all elements are polynomials over $\mathcal{R}_q = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ where we always assume implicit reduction modulo q and reduction modulo $x^n + 1$ and only allow parameters for which it holds that $1 \equiv q \mod 2n$ for q being a prime and n being a power of two. For efficiency, we make explicit use of the NTT in a way that has been previously described in [38, 47]. For efficiency we transmit and store keys and some ciphertexts in the NTT domain. Note that for the discussion of the masking scheme it is sometimes not relevant whether polynomials are stored in NTT format or whether the NTT is used at all (other options would be schoolbook or Karatsuba) and thus we sometimes omit the NTT notations to simplify the presentation. The public key $(\mathbf{p} = \mathbf{r}_1 - \mathbf{ar}_2, \mathbf{a})$ is an ring-LWE sample and an attacker trying to extract the secret key basically has to solve the search version of the ring-LWE problem [33]. In earlier works [11, 24] RLWE.CPA , or derived key-exchange schemes, was usually instantiated with a (high-precision) discrete Gaussian distribution with parameter σ. However, newer results show that security can also be achieved with distributions that are close to a discrete Gaussian. Examples are the binomial distribution [3, 13], a fixed distribution [12], a binary distribution [15], or a uniform distribution [4, 26]. We define SampleNoisePoly() to be a function that samples a polynomial in $\mathcal{R}_q$ with coefficients coming from a binomial distribution with parameter k where each coefficient is sampled independently as $\sum_{i=0}^{k-1} b_i - b'_i$, where the $b_i, b'_i \in \{0,1\}$ are uniform independent bits.[1] The binomial distribution is centered with a zero mean, has variance $k/2$, and gives a standard deviation of $\varsigma = \sqrt{k/2}$. For distributions that roughly follow a discrete Gaussian the standard deviation ς can be considered as the most important measure when describing and comparing security levels for

[1] In [3] the definition of the binomial distribution contains a typo in which the sum goes from zero to k.

ring-LWE. A uniformly random polynomial is sampled by SampleUniformPoly() and we decided to include $\tilde{\mathbf{a}}$ in the public key for simplification. Note that it would be possible to generate the secret key or $\tilde{\mathbf{a}}$ from a seed of 256-bits (or to choose $\tilde{\mathbf{a}}$ as a global constant; see [3] for a discussion). Additionally, the secret key $\tilde{\mathbf{r}}_2$ could be generated from a seed or stored in normal domain and efficiently encoded as it is not distributed uniformly but roughly follows a discrete Gaussian (see [39, 49]). However, for comparability and maintainability, we leave these straightforward optimizations and trade-offs as future work as they are not essential for our use-case. For successful decryption knowledge of the secret key $\mathbf{r}_2$ is required. Otherwise, the large term $\mathbf{ae}_1\mathbf{r}_2$ cannot be eliminated when computing $\mathbf{c}_1\mathbf{r}_2+\mathbf{c}_2$. An encoding of the n-bit message m is necessary as some small noise (i.e., $\mathbf{e} = \mathbf{e}_1\mathbf{r}_1 + \mathbf{e}_2\mathbf{r}_2 + \mathbf{e}_3$) is still present after calculating $\mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2$ and would prohibit the retrieval of the message after decryption. This also shows why the noise distribution is chosen to be rather small—a too big noise level would make reliable decoding impossible. Thus, to allow the extraction of the message despite the noise during decryption RLWE.CPA requires (as a minimum) a simple message encoding. We replace the standard threshold encoding and decoding functions with a variant that encodes one message bit into four coefficients [38]. The encoding function used in $\mathsf{RLWE.CPA}_{\text{enc}}^{\text{NTT}}$ is defined as $\mathsf{Encode}(m \in \{0,1\}^{n/4}) = \sum_{i=0}^{n-1} m[\lfloor i/4 \rfloor] \cdot \frac{q}{2} \cdot x^i$ (where $m[i]$ denotes the i-th bit of m). The decoding function used in $\mathsf{RLWE.CPA}_{\text{dec}}^{\text{NTT}}$ takes four coefficients $z_1, z_2, z_3, z_4 \in [-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$ as input that carry one bit of the message. $\mathsf{Decode}(z_1, z_2, z_3, z_4)$ is defined to return 1 if $|z_1|+|z_2|+|z_3|+|z_4| < q$ and 0 otherwise.

2.2.3 Side-Channel Attacks and Countermeasures

In this section we review side-channel attacks and countermeasures.

Timing Attacks When implementing cryptographic algorithms, the developer has to make sure that the execution time is independent of the secret data that is processed. Otherwise an attacker might be able to exploit the information about the execution time. Such attacks should not only be considered for embedded devices for which the attacker has physical access to, but also remote timing attacks are a threat that must be considered as shown by Brumley and Boneh [14]. Timing information can be leaked by conditional branches, instructions with non-constant execution time, and memory accesses that trigger cache hits or misses [8].

Differential Power Analysis Introduced in 1998 by Kocher et al. [31], DPA needs many power traces and one analyzes the set of traces with statistical methods. When performing DPA an attacker does not attack the whole key at once, but only a part, e.g., one byte. A DPA is divided in an online phase and an offline phase. During the online phase, the attacker runs a vast amount of executions of the algorithm to be attacked with different inputs and measures the power consumption of the

target device during each run. DPA requires a leakage model that is a prediction of the power consumption. Some leakage models are rather simple. For example, the Hamming weight model is based on the observation that the Hamming weight of a value that is stored in a register influences the power consumption. During the offline phase, the attacker guesses the key byte and computes the intermediate value that he considers suitable to apply the power model to. Depending on the power model and the intermediate value, she assigns the corresponding power trace to one of the two sets where one contains power traces with high predicted power consumption and one set contains traces with low prediction power consumption. For all power traces, the attacker stores the difference of the means of the sets. If the attack worked, the correct key guess has a much higher difference of means than the other guesses.

Hiding Hiding countermeasures are applied to raise the difficulty for an attacker to detect sensitive information in a set of power traces. This can be achieved by introducing additional noise or by trying to equalize the power consumption of all operations. The first approach can be achieved by other computations that are executed in parallel or by shuffling the order of operations. For hardware implementations one can even instantiate dedicated noise generators to randomize the power consumption. If shuffling is applied an attacker needs to perform an extra alignment step before analyzing the power traces. Otherwise the number of required power traces drastically increases. The second approach is more suitable for hardware implementations as in microcontrollers the developer has only limited influence on the power consumption of an instruction and only one instruction can be executed in parallel (except the microcontroller features SIMD instructions).

Masking The idea behind masking is to split a secret value into several shares. The secret value can only be reconstructed with the knowledge of all shares. The splitting of the secret value can be performed in a Boolean way or in an arithmetic way. Boolean masking means that the XOR-sum of all shares results in the secret value and arithmetic masking means that the arithmetic sum or difference of the shares results in the secret value. There are conversion approaches to switch between arithmetic and Boolean masking [18]. The major advantage of masking schemes is that they allow to prove the side-channel security of an algorithm. Nevertheless, there are still implementation challenges that have to be taken care of. Otherwise, a provably secure algorithm might still have a side-channel leakage. To achieve higher-order security, it is necessary to split the secret value into more shares.

2.3 CCA2 Conversion and Masking

In this section we describe how ring-LWE can be made resilient to CCA and side-channel attacks using the Targhi–Unruh variant of the Fujisaki–Okamoto [22, 54] (FO) transformation and our masking scheme.

2.3.1 CCA2 Conversion for RLWE.CPA

In this work we use the Fujisaki–Okamoto [22] transformation to enable a semantically secured encryption with respect to adaptive chosen-ciphertext attack (CCA2). For this transformation, Peikert came to the conclusion [37] that a passively secured encryption scheme should be converted into an actively secured one (based on the random oracle model; assuming adaptive attacks for CCA2). For this transformation, two random oracles $G : \{0,1\}^L \rightarrow \{0,1\}^l$ and $H : \{0,1\}^{L+l} \rightarrow \{0,1\}^\lambda$ are required. Targhi and Unruh pointed out that a third random oracle $H' : \{0,1\}^L \rightarrow \{0,1\}^l$ is necessary for the quantum security of the transformation [54]. The parameter L determines the size of the message to be encrypted, l the length of the input to ring-LWE encryption, and λ the length of the seed for the pseudo-random number generator (PRNG). In our implementation, the parameters L, l, and λ are set to 256 and we define $\mathsf{RLWE.CCA}_{\mathrm{enc}}^{\mathrm{NTT}}$ and $\mathsf{RLWE.CCA}_{\mathrm{dec}}^{\mathrm{NTT}}$ as follows:

- $\mathsf{RLWE.CCA}_{\mathrm{enc}}^{\mathrm{NTT}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, m_{cca} \in \{0,1\}^L)$:
  Let $(\tilde{\mathbf{c}_1}, \mathbf{c}_2) = \mathsf{RLWE.CPA}_{\mathrm{enc}}^{\mathrm{NTT}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \nu; H(\nu||m_{cca}))$, where $\nu \in \{0,1\}^L$ is a nonce and $H(\nu||m_{cca})$ seeds the PRNG of $\mathsf{RLWE.CPA}_{\mathrm{enc}}^{\mathrm{NTT}}$. Compute $c_3 = G(\nu) \oplus m_{cca}$ as well as $c_4 = H'(\nu)$ and output $(\tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)$.
- $\mathsf{RLWE.CCA}_{\mathrm{dec}}^{\mathrm{NTT}}(\tilde{\mathbf{r}}_2, \tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)$:
  Compute $\nu' = m_{cpa} = \mathsf{RLWE.CPA}_{\mathrm{dec}}^{\mathrm{NTT}}(\tilde{\mathbf{r}}_2, \tilde{\mathbf{c}}_1, \mathbf{c}_2)$, $m_{cca} = G(\nu') \oplus c_3$, $(\tilde{\mathbf{c}}_1^*, \mathbf{c}_2^*) = \mathsf{RLWE.CPA}_{\mathrm{enc}}^{\mathrm{NTT}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \nu'; H(\nu'||m_{cca}))$, and $c_4^* = H(\nu')$. Check if $(\tilde{\mathbf{c}_1}, \mathbf{c}_2) \stackrel{?}{=} (\tilde{\mathbf{c}}_1^*, \mathbf{c}_2^*)$ and $c_4 \stackrel{?}{=} c_4^*$. If so, output m_{cca}, otherwise output $fail$.

Using this transformation and our chosen parameters we obtain a theoretical public-key size of $|(\tilde{\mathbf{a}}, \tilde{\mathbf{p}})| = 2n\lceil \log_2(q) \rceil = 2 \cdot 1024 \cdot 14 = 28{,}672$ bits (3584 bytes) and a theoretical ciphertext size of $|(\tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)| = 2n\lceil \log_2(q) \rceil + 2l = 29{,}184$ bits (3648 bytes). The secret key is $|\tilde{\mathbf{r}}_2| = n\lceil \log_2(q) \rceil = 14{,}336$ bits (1792 bytes).

2.3.2 Masked CCA2-Secured Ring-LWE Decryption

To achieve side-channel resistance, it is necessary to mask all vulnerable modules of the CCA2-secured decryption. As depicted in Fig. 2.1 in bold notation, these modules are $\mathsf{RLWE.CPA}_{\mathrm{dec}}^{\mathrm{NTT}}$, G, H, H', and $\mathsf{RLWE.CPA}_{\mathrm{enc}}^{\mathrm{NTT}}$, and the two comparisons. Note that it is not sufficient to only protect $\mathsf{RLWE.CPA}_{\mathrm{dec}}^{\mathrm{NTT}}$, because in a chosen-ciphertext setting an adversary can target the unmasked output of $\mathsf{RLWE.CPA}_{\mathrm{dec}}^{\mathrm{NTT}}$ (see Appendix B of [44]) to recover the secret key. This attack trivially extends to any other intermediate variable which depends on m_{cpa}. A DPA-adversary would keep $\mathbf{c}_1, \mathbf{c}_2$ constant while varying c_3 and c_4. This way it is possible to derive hypothetical values for every other module following $\mathsf{RLWE.CPA}_{\mathrm{dec}}^{\mathrm{NTT}}$ depending on a guess for m_{cpa} (which only depends on one coefficient of $\mathbf{r}_2$ in a chosen-ciphertext setting). Therefore, even the final comparison needs to be protected against a side-channel adversary.

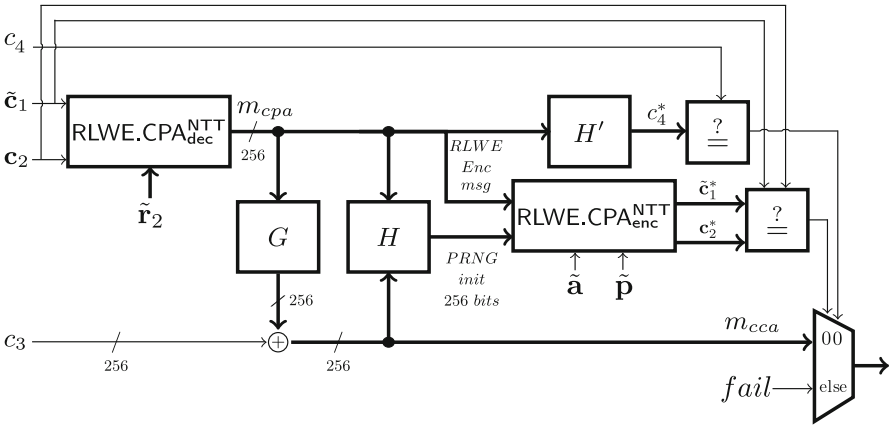

Fig. 2.1 CCA2-secured decryption

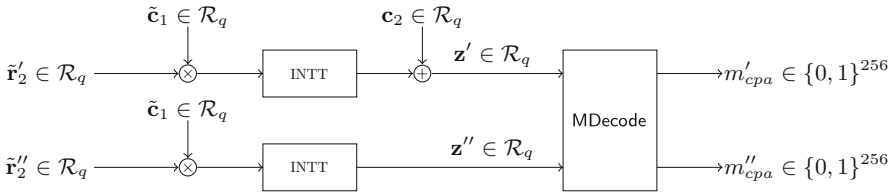

Fig. 2.2 Proposed masking scheme for ring-LWE decryption

In the following, we analyze the first-order security of each module separately in the common probing model [28]. To this end, we show that an attacker, who can probe one intermediate variable of the computation, cannot derive any secret information. This notion is equivalent to showing that each intermediate variable follows a distribution independent of any sensitive variable, i.e., the secret key $\mathbf{r}_2$. For one probe it is indeed sufficient to analyze each module separately, if the input and output distributions between the modules are consistent. Therefore, 1-probing security with correct input distributions for each module implies 1-probing security of the complete masked CCA2-secured decryption. However, for more probes (i.e., 2-probing security) this approach would not cover every possible attack vector and a more sophisticated analysis has to be utilized [7].

Ring-LWE Decryption As mentioned in Sect. 2.1, the masking schemes of the ring-LWE decryption from works like [43, 46] and [45] suffer from a higher failure probability and slower performance. Therefore, we present a new approach which avoids the aforementioned problems and still provides side-channel protection. Figure 2.2 shows the basic structure of our masked ring-LWE decryption. For the initial multiplications, additions, and INTTs we rely on a simple randomized sharing of $\mathbf{r}_2 = \mathbf{r}_2' + \mathbf{r}_2''$ with $\mathbf{r}_2' \xleftarrow{\$} \mathcal{R}_q$ similar to [43, 46]. Given the

linearity of the operations, it is easily possible to perform these computations on each share separately. However, this approach does not work for the final Decode. In [43, 46], the authors proposed to use a rather complex decoder for the arithmetically masked shares instead. To increase efficiency, we rely on a new approach MDecode which first transforms the arithmetic shares to Boolean shares and then performs the decoding. With this approach, we can avoid the costly arithmetically masked decoder and the additional error of the scheme from [45].

Correctness To show the correctness of this scheme, we first denote the outputs of the INTT operations as $\mathbf{z}'$ and $\mathbf{z}''$ with $\mathbf{z} = \mathbf{z}' + \mathbf{z}''$. Showing that this relation holds is trivial, since the INTT is linear and the scheme is identical to [43, 46] up to this point. Instead, we show that $\mathsf{MDecode}(\mathbf{z}', \mathbf{z}'') = (m'_{cpa}, m''_{cpa})$ with $m_{cpa} = m'_{cpa} \oplus m''_{cpa}$. To this end, we start by describing how an arithmetic-to-Boolean (A2B) transformation [17, 19, 21, 25, 56] can be used to easily decode one shared coefficient of $\mathbf{z}$. Then we demonstrate a solution to efficiently adjust the approach to our encoding scheme, i.e., four coefficients of $\mathbf{z}$ for one bit of m_{cpa}.

In our basic example, we assume the arithmetic shares (x_1, x_2) with

$$x_1 + x_2 \pmod{q} = x = m \cdot \lfloor \frac{q}{2} \rfloor + e$$

for some error e and want to recover (m_1, m_2) with $m_1 \oplus m_2 = m$ without leaking sensitive information. Our solution to this problem is based on the observation that a sharing of the most significant bit can be easily extracted from Boolean shares, while it is hard for arithmetic shares. However, we cannot straightforwardly apply an A2B transformation to (x_1, x_2) as all A2B algorithms work with arithmetic shares which are computed modulo a power of two.

Therefore, we propose to first transform (x_1, x_2) to the shares (y_1, y_2) with $y_1 + y_2 \mod 2^{15} = x$ given that 2^{15} is the second-next-larger power of two for $q = 12{,}289$. This process is shown in Algorithm 1 where every operation is done mod 2^{bits}, `A2B` denotes an arithmetic-to-Boolean transformation, and `MSB` returns the most significant bit of the input. In the algorithm, we first sample a random 15-bit value y_1 and reshare the input shares mod 2^{bits}. However, in some cases this does not result in a correct sharing as in Line 3 the shares are

$$y_1 + y_2 \mod 2^{bits} = x + q \cdot carry,$$

where the $carry$ is set if $x_1 + x_2 \geq q$. To adjust this, we compute $carry$ and subtract $q \cdot carry$ from (y_1, y_2) in a secured fashion. First, we compute $z_1 \leftarrow y_1 - q \mod 2^{bits}$. By doing this, we create the following relation for the most significant bit of $z_1 + y_2 \mod 2^{bits}$:

$$\mathtt{MSB}(z_1 + y_2 \mod 2^{bits}) = \begin{cases} 0 & x_1 + x_2 \geq q \\ 1 & x_1 + x_2 < q \end{cases},$$

if $bits \geq \log_2(2q)$. Therefore, we have $\texttt{MSB}(z_1 + y_2 \bmod 2^{bits}) \oplus 1 = carry$. Then we use the `A2B` algorithm by Debraize [21], so that we can apply `MSB` to each of the output shares separately. The only remaining step now is to subtract $q \cdot carry$ from (y_1, y_2). This is achieved using the shares $k_1 \oplus k_2 = carry$ and the relation $k_1 \oplus k_2 = k_1 + k_2 - 2k_1k_2$ as follows:

$$\begin{aligned} y_1 - (k_1 \oplus k_2)q = \; & y1 - k_1q - k_2q + 2k_1k_2q \\ = \; & y_1 - k_1q - k_2q + 2(k_1' + k_1'')(k_2' + k_2'')q \\ = \; & y_1 - k_1q - k_2q + 2k_1'k_2'q + 2k_1'k_2''q + 2k_1''k_2'q + 2k_1''k_2''q. \end{aligned}$$

Since (k_1, k_2) is not completely independent of (y_1, y_2) for some `A2B`, we include a random value r in the computation of the sum in Algorithm 1.

Although the output (y_1, y_2) of `TransformPower2` fulfills the desired property of $y_1 + y_2 \bmod 2^{15} = x$ and could be easily transformed to (y_1', y_2') with $y_1 \oplus y_2 = x$, this is not sufficient to recover m. Some additional steps are necessary to perform a successful decoding. These steps are depicted in Fig. 2.3. Each circle shows the distributions of the unshared values for a specific value of m ($m = 0$ is thick, $m = 1$ is dashed) after each step, e.g., the first circle in the upper-left corner shows the distributions for the original x where the values of x for $m = 0$ (resp. $m = 1$) are grouped around the mean of zero (resp. $\frac{q}{2}$). In the first step, we subtract $\frac{q}{4}$ from (x_1, x_2). This way no distribution is spread over the modulo border, which would cause problems for the transformation to 15 bits. After the transformation is done, we subtract $\frac{q}{2}$ from the result to create the following relation for the new shares (y_1, y_2):

Algorithm 1 `TransformPower2`

Input: $x_1, x_2, bits$
Output: y_1, y_2
1: $y_1 \xleftarrow{\$} \{0,1\}^{bits}$
2: $y_2 \leftarrow x_1 - y_1$
3: $y_2 \leftarrow y_2 + x2$
4: $z_1 \leftarrow y_1 - q$
5: $[z_1, z_2] \leftarrow \texttt{A2B}(z_1, y_2)$
6: $k_1 \leftarrow \texttt{MSB}(z_1) \oplus 1$
7: $k_2 \leftarrow \texttt{MSB}(z_2)$
8: $k_1' \xleftarrow{\$} \{0,1\}^{bits}$
9: $k_1'' \leftarrow k_1 - k_1'$
10: $k_2' \xleftarrow{\$} \{0,1\}^{bits}$
11: $k_2'' \leftarrow k_2 - k_2'$
12: $r \xleftarrow{\$} \{0,1\}^{bits}$
13: $y_1 = (((((((r + y_1) - k_1q) - k_2q) + 2k_1'k_2'q) + 2k_1'k_2''q) + 2k_1''k_2'q) + 2k_1''k_2''q)$
14: $y_2 = y_2 - r$

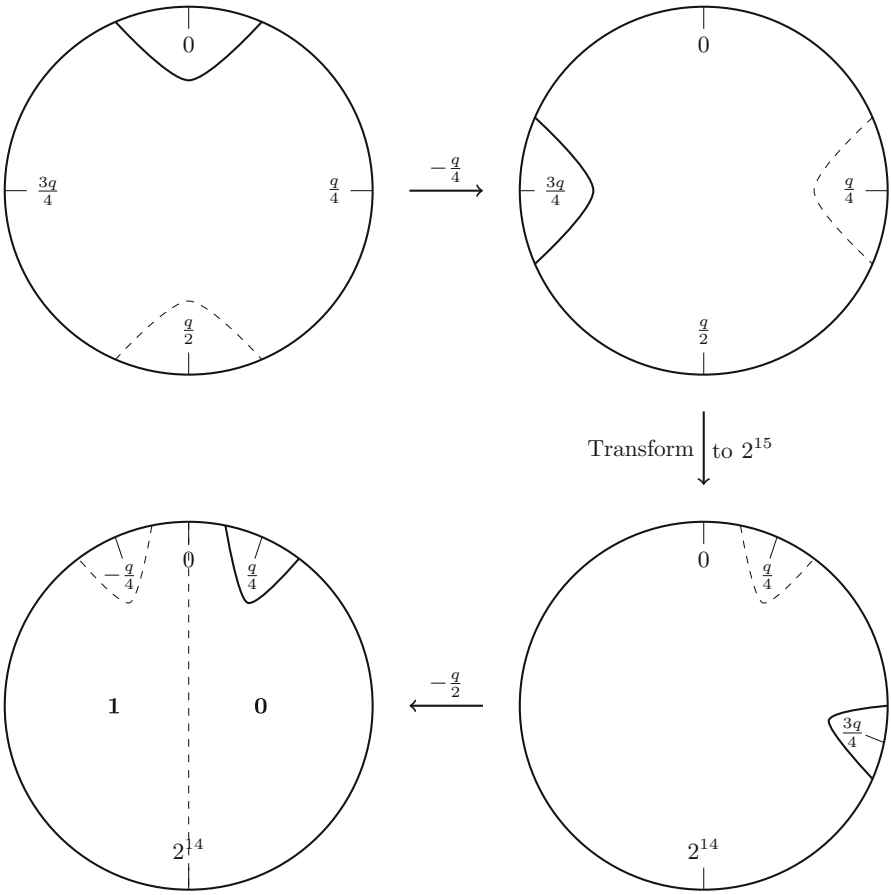

Fig. 2.3 First three steps when decoding one coefficient

$$\texttt{MSB}(y_1 + y_2 \bmod 2^{bits}) = \begin{cases} 0 & m = 0 \\ 1 & m = 1 \end{cases},$$

as the distributions are equally distant to zero which prevents an increase in the error probability of the decoding. In the last step, we again perform an A2B transformation $\texttt{A2B}(y_1, y_2) = (y_1', y_2')$ to easily extract a sharing of m with $\texttt{MSB}(y_1') \oplus \texttt{MSB}(y_2') = m_1 \oplus m_2 = m$.

For four related coefficients, one possible approach is to perform the aforementioned masked decoding for each coefficient separately and then combine them via a masked majority function. However, a more efficient solution is described in Algorithm 2, where (a_1, a_2), (b_1, b_2), (c_1, c_2), and (d_1, d_2) are four related shared coefficients (i.e., encode the same m). Our main idea is to combine the coefficients before the final A2B. To perform this combination without losing information and

Algorithm 2 MDecode

Input: $a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2$
Output: m_1, m_2
1: $a_1 \leftarrow a_1 - \lfloor \frac{q}{4} \rfloor$
2: $b_1 \leftarrow b_1 - \lfloor \frac{q}{4} \rfloor$
3: $c_1 \leftarrow c_1 - \lfloor \frac{q}{4} \rfloor$
4: $d_1 \leftarrow d_1 - \lfloor \frac{q}{4} \rfloor$
5: $[a_1, a_2] \leftarrow$ `TransformPower2`$(a_1, a_2, 16)$
6: $[b_1, b_2] \leftarrow$ `TransformPower2`$(b_1, b_2, 16)$
7: $[c_1, c_2] \leftarrow$ `TransformPower2`$(c_1, c_2, 16)$
8: $[d_1, d_2] \leftarrow$ `TransformPower2`$(d_1, d_2, 16)$
9: $e_1 \leftarrow a_1 + b_1 + c_1 + d_1$
10: $e_2 \leftarrow a_2 + b_2 + c_2 + d_2$
11: $e_1 \leftarrow e_1 - 2q$
12: $[e_1, e_2] \leftarrow$ `A2B`(e_1, e_2)
13: $m_1 =$ `MSB`(e_1)
14: $m_2 =$ `MSB`(e_2)

keeping the same error probability, we have to increase the number of bits for `TransformPower2` to $bits \geq \log_2(2 \cdot 4 \cdot \frac{q}{2})$, i.e., 16 for $q = 12{,}289$. After the transformation, we can easily sum the coefficients sharewise. We also have to adjust the last subtraction to $2q$. If no error has occurred (i.e., all coefficients encode the same m), there are two distributions with means $2^{16} - q$ and $+q$ and (m_1, m_2) can be easily recovered with a final `A2B`. In this way, we save three calls to `A2B` compared to the naive majority approach.

Security Analysis We analyze the security of Algorithms 1 and 2 by showing that each intermediate variable follows a distribution independent of any sensitive variable. For `TransformPower2` this is formalized in the following lemma.

Lemma 2.1 *When $x_1, x_2 \in \mathbb{Z}_q$ are a uniform sharing of $x = x_1 + x_2 \pmod q$ and $y_1, k_1', k_2', r \in \{0,1\}^{bits}$ are uniformly and independently distributed in their respective value spaces, all intermediate variables in Algorithm 1 have a distribution independent of the sensitive variable x.*

Proof For the proof, we analyze the distributions of the variables of each line from Algorithm 1 and show that their distributions are independent of the sensitive variable x.

- *Lines 2, 3*: Since y_1 is a random value in $\{0,1\}^{bits}$, $(x_1 - y_1)$ is also a random variable following a distribution independent of x. The same applies to $(x_1 - y_1) + x_2 = x - y_1$.
- *Line 4*: A constant value is subtracted from a random value in $\{0,1\}^{bits}$ which does not leak about x.
- *Line 5*: The security strongly depends on the chosen transformation algorithm. In our implementation, we use the algorithm from [21] and refer the interested reader to their proof of security.

- *Lines 6, 7, 9, 11*: Each of these lines operates on only one of the shares. Therefore, each of them follows a distribution independent of x assuming `A2B` to be secured.
- *Line 13*: The first operand of the sum is the random value $r \in \{0,1\}^{bits}$. Therefore, all following operations are perfectly masked by r and follow a distribution independent of x.
- *Line 14*: A random value is subtracted from only one share. Therefore, the result does not leak about x.

As shown above, the distribution of every intermediate variable of Algorithm 1 is independent of the sensitive variable x. The output shares y_1 and y_2 with $x = y_1 + y_2 \mod 2^{bits}$ are both uniformly distributed in $\{0,1\}^{bits}$.

For MDecode, the security properties are formalized in the following lemma.

Lemma 2.2 *When $a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 \in \mathbb{Z}_q$ are uniform shares $a = a_1 + a_2 \pmod q$, $b = b_1 + b_2 \pmod q$, $c = c_1 + c_2 \pmod q$, $d = d_1 + d_2 \pmod q$ which are independent of each other, all intermediate variables in Algorithm 2 have a distribution independent of the sensitive variables a, b, c, d, and m.*

Proof For the proof, we analyze the distributions of the variables of each line from Algorithm 2 and show that their distributions are independent of the sensitive variables.

- *Lines 1–4*: A constant value is subtracted from only one share. If the input sharings are uniform, the result is still a uniform sharing independent of the sensitive variables.
- *Lines 5–8*: The security depends on the security of `TransformPower2` which is analyzed in the previous lemma.
- *Lines 9, 10*: Assuming the output sharings of the four calls to `TransformPower2` are still uniform and independent, processing only one share of each sharing is always independent of the sensitive variables.
- *Line 11*: (e_1, e_2) are a uniform sharing of $e = a + b + c + d$. Since only one share is processed, the result is independent of the sensitive variables.
- *Line 12*: Again the security depends on the chosen algorithm for `A2B`.
- *Lines 13, 14*: Each of these lines operates on only one of the shares. Therefore, each of them follows a distribution independent of the sensitive variables assuming `A2B` to be secured.

As shown above, the distribution of every intermediate variable of Algorithm 2 is independent of the sensitive variables a, b, c, d, and m. The output shares m_1 and m_2 with $m = m_1 \oplus m_2$ are both uniformly distributed in $\{0,1\}$.

***G*, *H*, and *H′* (SHAKE)** We choose to instantiate G, H, and H' with the commonly used extendable-output function SHAKE that is based on the KECCAK algorithm [10] and apply the masking scheme presented in [9]. Therefore, we do not include the security analysis of this module and instead refer the reader to the original publications. We use a different initialization vector for each instantiation of the random oracles to make G, H, and H' distinct from each other.

Ring-LWE Encryption For the masked $\mathsf{RLWE.CPA}_{\mathrm{enc}}^{\mathrm{NTT}}$ (i.e., the re-encryption in $\mathsf{RLWE.CCA}_{\mathrm{dec}}^{\mathrm{NTT}}$), every input or internally PRNG-generated variable is sensitive (i.e., $m_{cpa}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$) since they can be used to recover the secret key $\mathbf{r}_2$ as detailed in the beginning of this section. Therefore, the computation of $\mathbf{c}_1$ and $\mathbf{c}_2$ is done in the shared domain. For the former this is trivial, since it only requires linear operations which can be performed on each input share separately as

$$\mathbf{c}'_1 = \mathbf{a} \cdot \mathbf{e}'_1 + \mathbf{e}'_2,$$
$$\mathbf{c}''_1 = \mathbf{a} \cdot \mathbf{e}''_1 + \mathbf{e}''_2.$$

Due to the simplicity of this computation we omit the security analysis.

For $\mathbf{c}_2$, however, we have to consider the rounding error from Encode to obtain the correct result, i.e., it is not sufficient to compute

$$\mathbf{c}'_2 = \mathbf{p} \cdot \mathbf{e}'_1 + \mathbf{e}'_3 + \mathsf{Encode}(m'_{cpa}),$$
$$\mathbf{c}''_2 = \mathbf{p} \cdot \mathbf{e}''_1 + \mathbf{e}''_3 + \mathsf{Encode}(m''_{cpa}).$$

In this equation, $m'_{cpa} \oplus m''_{cpa} = m_{cpa}$. Since our modulus q is odd and therefore $2\lfloor \frac{q}{2} \rfloor \neq q$, we have to adjust this operation so that the correct result is computed, i.e., the result of the re-encryption has to be exactly the same as the result of the original encryption. The naive approach would be to multiply one of the intermediate results, e.g., $\mathbf{c}'_2$ (without the message), by 2, encode the shares of m_{cpa} as $\{0, q\}$, perform two additions modulo $2q$, and divide the result by 2. While this approach indeed yields the correct result, it introduces an easily detectable side-channel leakage as the last bit of the intermediate results before the division is always set to 1 if and only if the unshared message bit is 1, i.e., q has been added exactly one time. Similarly, the last bit is always set to 0 if and only if the unshared message bit is 0. We cannot apply the technique described in [37] as adding a random bit yields a different result if the value that bit is added to is odd. In the CCA2 setting, it is required that both the original encryption and the re-encryption output exactly the same result and thus even a single bit error is not tolerable.

We thus decided to only return a false result in case both shares, m'_{cpa} and m''_{cpa}, have the value 1. In this case, the `floor` operation cuts off $\frac{1}{2}$ two times and thus the result is off by one. To get the correct result, we have to add m'_{cpa} `AND` m''_{cpa}. Obviously, we cannot compute this multiplication of the shares directly without leakage. Thus, we split the shares into subshares.

$$m'_{cpa} = \mathbf{m}'_{\mathbf{cpa},\mathbf{1}} + \mathbf{m}'_{\mathbf{cpa},\mathbf{2}}$$
$$m''_{cpa} = \mathbf{m}''_{\mathbf{cpa},\mathbf{1}} + \mathbf{m}''_{\mathbf{cpa},\mathbf{2}}$$

Notice that for this calculation m'_{cpa} and m''_{cpa} are implicitly treated as polynomials in $\mathcal{R}_q$ and not as bit vectors. For simplicity, we assume in this description

that one bit is encoded into one coefficient but this approach trivially generalizes to multi-coefficient encodings as well. As a consequence of the splitting into shares, we have to compute $(\mathbf{m}'_{\mathbf{cpa},\mathbf{1}}+\mathbf{m}'_{\mathbf{cpa},\mathbf{2}})\circ(\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}+\mathbf{m}''_{\mathbf{cpa},\mathbf{2}})$ instead of m'_{cpa} AND m''_{cpa}. To obtain the correct result, we compute

$$\begin{aligned}\mathbf{c}'_2 = \; & (\mathbf{p}\cdot\mathbf{e}'_1+\mathbf{e}'_3+\mathsf{Encode}(m'_{cpa})) \\ & +\mathbf{m}'_{\mathbf{cpa},\mathbf{1}}\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}+\mathbf{m}'_{\mathbf{cpa},\mathbf{1}}\mathbf{m}''_{\mathbf{cpa},\mathbf{2}}+\mathbf{m}'_{\mathbf{cpa},\mathbf{2}}\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}+\mathbf{m}'_{\mathbf{cpa},\mathbf{2}}\mathbf{m}''_{\mathbf{cpa},\mathbf{2}}\end{aligned}$$

Note that the term $\mathbf{p}\cdot\mathbf{e}'_1+\mathbf{e}'_3$ provides the randomness to protect the masked AND computation akin to Trichina's masked AND [55]. Therefore, the order of operations in the computation of $\mathbf{c}'_2$ is important for the security. Our complete masked re-encryption is shown in Algorithm 3.

Lemma 2.3 *When* $\mathbf{e}'_1+\mathbf{e}''_1=\mathbf{e}_1\in\mathcal{R}_q$, $\mathbf{e}'_3+\mathbf{e}''_3=\mathbf{e}_3\in\mathcal{R}_q$, $m'_{cpa}+m''_{cpa}=m_{cpa}\in\{0,1\}^{n/4}$ *are uniform, independent shared representations of the sensitive input variables and* $\mathbf{m}'_{\mathbf{cpa},\mathbf{1}},\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}\in\mathcal{R}_q$ *are uniform and independent random variables, all intermediate variables in Algorithm 3 have a distribution independent of the sensitive variables* m_{cpa}, $\mathbf{e}_1$, *and* $\mathbf{e}_3$.

Proof For the proof, we analyze the distributions of the variables of each line from Algorithm 3 and show that they are independent of the sensitive variables m_{cpa}, $\mathbf{e}_1$, and , $\mathbf{e}_3$.

- *Lines 1, 2*: Each of these lines only uses one of the shares and is therefore independent of the sensitive variables. The shared representation of the error vectors is independent of the shared representation of m_{cpa} due to the mask refresh inside the shared sampler.
- *Lines 3, 4*: $m'_{cpa,1}$ (resp. $m''_{cpa,1}$) are new random masks that are used to mask the shares of m_{cpa}. Since only one share of m_{cpa} is involved in each line, the result is still independent of m_{cpa}.

Algorithm 3 Masked ring-LWE encryption

Input: $\mathbf{p}, \mathbf{e}'_1, \mathbf{e}'_3, \mathbf{e}''_1, \mathbf{e}''_3, m'_{cpa}, m''_{cpa}, \mathbf{m}'_{\mathbf{cpa},\mathbf{1}}, \mathbf{m}''_{\mathbf{cpa},\mathbf{1}}$
Output: $\mathbf{c}'_2, \mathbf{c}''_2$
1: $\mathbf{c}'_2 \leftarrow \mathbf{p}\cdot\mathbf{e}'_1+\mathbf{e}'_3+\mathsf{Encode}(m'_{cpa})$
2: $\mathbf{c}''_2 \leftarrow \mathbf{p}\cdot\mathbf{e}''_1+\mathbf{e}''_3+\mathsf{Encode}(m''_{cpa})$
3: $\mathbf{m}'_{\mathbf{cpa},\mathbf{2}} \leftarrow m'_{cpa}-\mathbf{m}'_{\mathbf{cpa},\mathbf{1}}$
4: $\mathbf{m}''_{\mathbf{cpa},\mathbf{2}} \leftarrow m''_{cpa}-\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}$
5: $\mathbf{t}_{11} \leftarrow \mathbf{m}'_{\mathbf{cpa},\mathbf{1}}\circ\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}$
6: $\mathbf{t}_{12} \leftarrow \mathbf{m}'_{\mathbf{cpa},\mathbf{1}}\circ\mathbf{m}''_{\mathbf{cpa},\mathbf{2}}$
7: $\mathbf{t}_{21} \leftarrow \mathbf{m}'_{\mathbf{cpa},\mathbf{2}}\circ\mathbf{m}''_{\mathbf{cpa},\mathbf{1}}$
8: $\mathbf{t}_{22} \leftarrow \mathbf{m}'_{\mathbf{cpa},\mathbf{2}}\circ\mathbf{m}''_{\mathbf{cpa},\mathbf{2}}$
9: $\mathbf{c}'_2 \leftarrow ((((\mathbf{c}'_2+\mathbf{t}_{11})+\mathbf{t}_{12})+\mathbf{t}_{21})+\mathbf{t}_{22})$

– *Lines 5, 6, 7, 8*: Both terms of each line are uniformly and independently distributed in $\mathcal{R}_q$. Therefore, the multiplication of these terms does not create a new dependency on m_{cpa} and the results can be easily simulated.
– *Line 9*: The term $(\mathbf{p} \cdot \mathbf{e}'_1 + \mathbf{e}'_3)$ is independent of m_{cpa} and therefore provides sufficient fresh randomness to protect the masked AND. Each intermediate variable of this line follows a uniform distribution in $\mathcal{R}_q$ independent of the sensitive variables m_{cpa}, $\mathbf{e}_1$, and $\mathbf{e}_3$.

As shown above, the distribution of every intermediate variable of Algorithm 3 is independent of the sensitive variables m_{cpa}, $\mathbf{e}_1$, and $\mathbf{e}_3$. Therefore, the aforementioned chosen-ciphertext attack is not possible.

Masked Binomial Sampler As detailed in the beginning of this section, the error vectors can be target for a chosen-ciphertext adversary in the side-channel setting. Therefore, we have to perform the sampling in a shared domain. We are using a binomial sampler that computes the Hamming weight of two bit vectors α and β and outputs the difference *out* of those Hamming weights. If we split α and β into two Boolean shares each, we can compute the output of the sampler as follows:

$$\begin{aligned} out &= \sum_{i=0}^{k-1}(\alpha_1[i] + \alpha_2[i] - 2\alpha_1[i]\alpha_2[i]) - \sum_{i=0}^{k-1}(\beta_1[i] + \beta_2[i] - 2\beta_1[i]\beta_2[i]) \\ &= \sum_{i=0}^{k-1}(\alpha_1[i] - \beta_1[i]) + \sum_{i=0}^{k-1}(\alpha_2[i] - \beta_2[i]) - 2\sum_{i=0}^{k-1}(\alpha_1[i]\alpha_2[i]) \\ &\quad + 2\sum_{i=0}^{k-1}(\beta_1[i]\beta_2[i]). \end{aligned}$$

Obviously, we cannot compute $\alpha_1[i]\alpha_2[i]$ and $\beta_1[i]\beta_2[i]$ directly. Instead, we compute them securely with the help of three random values $X, Y, Z \in [0, q-1]$ as shown in Algorithm 4.

Lemma 2.4 *When $\alpha_2 \in \{0,1\}^k$ with $\alpha_1 \oplus \alpha_2 = \alpha$, $\beta_2 \in \{0,1\}^k$ with $\beta_1 \oplus \beta_2 = \beta$, $X \in [0, q-1]$, $Y \in [0, q-1]$, and $Z \in [0, q-1]$ are uniformly and independently distributed in their respective value space, all intermediate variables in Algorithm 4 have a distribution independent of the sensitive unshared input variables α and β.*

Proof For the proof, we analyze the distributions of the variables of each line from Algorithm 4 and show that their distributions are independent of the sensitive variables α and β.

– *Lines 5, 6*: Only one share is used in each of the two operations. Therefore, the result is independent of the unshared values α and β.
– *Lines 8–13*: The proof works analogous to the proof for Lines 5–9 of Lemma 2.3.

Algorithm 4 Masked binomial sampler

Input: $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \{0,1\}^k$ with $\alpha_1 \oplus \alpha_2 = \alpha$ and $\beta_1 \oplus \beta_2 = \beta$
Output: out_1, out_2 with $(out_1 + out_2) \bmod q$ binomial distributed

$i \leftarrow 0$
$out_1 \leftarrow 0$
$out_2 \leftarrow 0$
for $i < k$ **do**
  $out_1 = out_1 + (\alpha_1[i] - \beta_1[i])$
  $out_2 = out_2 + (\alpha_2[i] - \beta_2[i])$
  $X \xleftarrow{\$} [0, q-1], Y \xleftarrow{\$} [0, q-1], Z \xleftarrow{\$} [0, q-1]$
  $\alpha_1'' = \alpha_1 - X$
  $\alpha_2'' = \alpha_2 - Y$
  $out_1 = out_1 - 2((((Z + XY) + X\alpha_2'') + \alpha_1''Y) + \alpha_1''\alpha_2'')$
  $\beta_1'' = \beta_1 - X$
  $\beta_2'' = \beta_2 - Y$
  $out_2 = out_2 + 2((((Z + XY) + X\beta_2'') + \beta_1''Y) + \beta_1''\beta_2'')$
  $i \leftarrow i + 1$
end for

As shown above, the distribution of every intermediate variable of Algorithm 4 is independent of the sensitive variables α and β. Therefore, it is not possible for an attacker, which can probe one value, to derive sensitive information. The output shares out_1 and out_2 with $out = out_1 + out_2$ are both uniformly distributed in $[0, q-1]$.

Masked PRNG The PRNG is also a possible target for a chosen-ciphertext adversary as noted before. Therefore, we used the already implemented masked version of SHAKE-128 to generate random numbers with a fixed seed.

2.3.3 Masked Comparison

It is further necessary to protect all comparisons against a side-channel adversary, since even $\tilde{\mathbf{c}}_1^*$, $\mathbf{c}_2^*$, and c_4^* can be used to distinguish $\tilde{\mathbf{r}}_2$. Since these values are shared, it is necessary to compute a function of both shares to compare them to the public and possibly adversary controlled values $\tilde{\mathbf{c}}_1$, $\mathbf{c}_2$, and c_4. To prevent leakage of the sensitive variables we introduce an additional hashing-step before the comparison. Using $\tilde{\mathbf{c}}_1$ as an example, we perform the comparison of $\tilde{\mathbf{c}}_1$ with the shared $\tilde{\mathbf{c}}_1^* = \tilde{\mathbf{c}}_1^{*\prime} + \tilde{\mathbf{c}}_1^{*\prime\prime}$ as provided in Algorithm 5. The correctness of our approach is easy to verify as

$$H''(\tilde{\mathbf{c}}_1^* - \tilde{\mathbf{c}}_1^{*\prime\prime}) \stackrel{?}{=} H''(\tilde{\mathbf{c}}_1^{*\prime})$$

$$\Leftrightarrow H''(\tilde{\mathbf{c}}_1^* - \tilde{\mathbf{c}}_1 + \tilde{\mathbf{c}}_1^{*\prime}) \stackrel{?}{=} H''(\tilde{\mathbf{c}}_1^{*\prime}).$$

Algorithm 5 Masked comparison of public $\tilde{\mathbf{c}}_1$ with internal $\tilde{\mathbf{c}}_1^*$

Input: $\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_1^{*\prime}, \tilde{\mathbf{c}}_1^{*\prime\prime}$
Output: eq
1: $\tilde{\mathbf{c}}_1^{*\prime} \leftarrow \tilde{\mathbf{c}}_1 - \tilde{\mathbf{c}}_1^{*\prime}$
2: $\tilde{\mathbf{c}}_1^{*\prime} \leftarrow H''(\tilde{\mathbf{c}}_1^{*\prime})$
3: $\tilde{\mathbf{c}}_1^{*\prime\prime} \leftarrow H''(\tilde{\mathbf{c}}_1^{*\prime\prime})$
4: $eq \leftarrow \tilde{\mathbf{c}}_1^{*\prime} \oplus \tilde{\mathbf{c}}_1^{*\prime\prime}$
5: $eq \leftarrow (eq == 0)$

Relying on the collision resistance of H'', this comparison is only true if the ciphertext is valid and thus $\tilde{\mathbf{c}}_1^{*\prime} = \mathbf{c}_1$.

Lemma 2.5 *When $\tilde{\mathbf{c}}_1^{*\prime} + \tilde{\mathbf{c}}_1^{*\prime\prime} = \tilde{\mathbf{c}}_1^* \in \mathcal{R}_q$ is a uniform, independent shared representation of the sensitive input variable $\tilde{\mathbf{c}}_1^*$ and H'' is a cryptographic hash function, every intermediate variable of Algorithm 5 is independent of the sensitive variable $\tilde{\mathbf{c}}_1^*$.*

Proof For the proof, we analyze the distributions of the variables of each line from Algorithm 5 and show that they are independent of the sensitive variable.

- *Lines 1–3*: Each line uses only one share of $\tilde{\mathbf{c}}_1^*$ and, therefore, the computation is independent of $\tilde{\mathbf{c}}_1^*$.
- *Line 4*: The adversary can probe $H''(\tilde{\mathbf{c}}_1^* - \tilde{\mathbf{c}_1} + \tilde{\mathbf{c}}_1^{*\prime}) \oplus H''(\tilde{\mathbf{c}}_1^{*\prime})$ which depends on both shares. However, we rely on the properties of H'' to break the linear relation between the shares and make a direct recovery of $\tilde{\mathbf{c}}_1^*$ impossible. Nevertheless, a computationally unbounded adversary would be able to distinguish the sensitive variable $\tilde{\mathbf{c}}_1^*$ by iterating over all possible $\tilde{\mathbf{c}}_1^{*\prime}$. Since $\tilde{\mathbf{c}}_1^{*\prime} \in \mathcal{R}_q$ this task is more complex than directly iterating over the whole key space of $\tilde{\mathbf{r}}_2$. Therefore, we do not consider this attack vector a viable threat. Furthermore, in the special case of $\tilde{\mathbf{c}}_1^* = \tilde{\mathbf{c}}_1$ the variable $\tilde{\mathbf{c}}_1^*$ is not sensitive.

However, it is only secure to have a function of both shares, because the comparison is always negative, i.e., eq is false, in a chosen-ciphertext setting. Therefore, the attacker does not gain additional knowledge from the output of the comparison. This does not apply to the comparison of $\mathbf{c}_2$ and c_4. In this case, the adversary can adaptively change $\mathbf{c}_2$ or c_4 without removing the sensitivity from m_{cpa} (which is not possible for $\tilde{\mathbf{c}}_1$) and use the output of $compare(\mathbf{c}_2^{*\prime}, \mathbf{c}_2^{*\prime\prime})$ (resp. $compare(c_4^{*\prime}, c_4^{*\prime\prime})$) to distinguish m_{cpa}. This problem can be solved by performing the other comparison (i.e., $\tilde{\mathbf{c}}_1$) beforehand and only if it returns true the other two comparisons (i.e., $\mathbf{c}_2$, c_4) are conducted. A timing-constant solution would be to perform dummy comparisons for $\mathbf{c}_2$ and c_4 in case the prior comparison failed. Furthermore, for these comparisons it is not even necessary to perform a masked comparison, since they are only ever done for valid $\tilde{\mathbf{c}}_1$ and in this setting m_{cpa} is not sensitive.

2.4 Implementation

To evaluate the performance of the CCA2-conversion and our masking scheme, we implemented the constructions on an ARM Cortex-M4F. Our evaluation platform is an STM32F4 DISCOVERY board with 1 Mbyte of flash memory, 192 kbyte of RAM, a floating-point unit (FPU), and a true random number generator (TRNG). In order to keep the running time constant and independent, we implemented critical components in assembly language. Furthermore, to prevent cache timing attacks we disabled the cache of the on-board flash memory by setting the `DCEN` bit of the `FLASH_ACR` register to zero.

We use SHAKE-128 as instantiation for all random oracles H, G, H', and H'' and use a different initialization vector for each of them. As the hashing plays a minor role in terms of performance, we selected the readable KECCAK implementation by Saarinen [48] as basis for our implementation as it allowed us to easily implement side-channel countermeasures. To achieve a constant running time we decided to implement the binomial sampler from [3] with $k = 8$. To sample the necessary randomness, we implemented a PRNG that is initialized with a 256-bit seed. For encryption, we generate this secret seed from the on-board TRNG and then use a PRNG to generate Gaussian noise. As we have to perform a re-encryption during the decryption that must sample the exact same values, we cannot use the TRNG for this purpose but have to initialize the PRNG with the same seed. We also use SHAKE-128 as PRNG.

For the implementation of polynomial arithmetic we need a high-performance and constant-time modular reduction to prevent remote timing attacks [40]. As a consequence, the implementation of the NTT and especially the three-instruction modular reduction from [20] is not suitable. It uses the `DIV` instruction, which has a data-dependent variable execution time that can reach from 2 to 12 clock cycles. Therefore, we implemented a Barrett reduction [6] using the FPU of the Cortex-M4F that takes 6 clock cycles and is timing-independent. De Clercq et al. [20] also present an optimized implementation of the NTT. They implement the NTT in assembly and also proposed an optimized memory access scheme. Their idea is to store two coefficients in one data word and being able to load/store both coefficients with the same instruction. Alkim et al. implemented the NTT as well as reported in [2]. By combining a Montgomery reduction with Barrett reduction, their NTT is considerably faster than the one from [20] and most importantly also has a constant execution time. We therefore embedded the NTT from [2] into our implementation.

A theoretically secure masking scheme can still show leakage in an actual implementation due to unconsidered effects inside the microarchitecture of the microcontroller. For instance, overriding a register that holds one share with the content of another register storing the other share will inevitably leak information. Similarly, one must avoid to load or store both shares from or to memory in consecutive instructions (or even the same instruction, e.g., load multiple `LDM`). Furthermore, carry bits can be a source of leakage. We carefully designed our implementation to not suffer from these problems. For operations that can be

performed on both shares independently (e.g., point-wise multiplication), this is easily achieved by executing the operations on all coefficients of one share first and only then do the operations for the coefficients of the other share. For operations that require both shares (i.e., Decode) hand-crafted assembly code is necessary.

2.5 Side-Channel Evaluation

Even though we provide proofs for most of our modules against one probe, practical first-order side-channel security is not automatically implied by that. Implementation errors can still negatively affect the resistance due to effects that are not included in the model [5]. Therefore, to extensively evaluate the security of our masked implementation, we performed basic side-channel experiments. Since our aim is to show first-order resistance, we rely on the commonly used t-test leakage detection methodology initially proposed in [16, 23]. We performed the test at first and second order. For bivariate second-order evaluation, we relied on the optimal centered product [41, 53] as the combination function.

We use a PicoScope 5203 with a sample rate of 125 MS/s to measure the power consumption at our STM32F4 Discovery board. To increase the measurement quality, we reduce the internal clock to 12 MHz and remove some capacitors from the PCB. The communication with the board is done over USART as the on-board USB interface causes additional noise in the power traces. Since the entirely masked decryption requires an extremely high number of clock cycles, we cannot easily perform a bivariate evaluation with our proposed method. Instead, we split the practical evaluation into the modules similar to the theoretical evaluation of Sect. 2.3.2. For first-order evaluation this is appropriate as noted in Sect. 2.3.2. However, for the bivariate second-order test we do not cover the scenario of two probes in different modules. Nevertheless, our goal is to show the existence of second-order leakage to verify our measurement setup and we found this for every module separately. For each module we took 100,000 measurements and performed the aforementioned tests. To further speed up the second-order evaluation, we adjusted the module to only process a small number of coefficients.

In our experiments, we perform the non-specific *fixed vs. random* t-test. To this end, we take two types of measurements. One with fixed input and one with random input. The t-statistic t is computed as

$$t = \frac{\mu_F - \mu_R}{\sqrt{\frac{\sigma_F^2}{n_F} + \frac{\sigma_R^2}{n_R}}},$$

where μ_F, σ_F^2, and n_F (resp. μ_R, σ_R^2, and n_R) denote the mean, variance, and number of measurements set with fixed input (resp. random input). If the value exceeds the threshold $|t| > 4.5$, the test has detected leakage. For more information, we refer the interested reader to further literature related to this side-channel evaluation methodology [51].

We measured the computation of the butterfly during the NTT for two coefficients, the addition of the two shares during the masked re-encryption as described in Sect. 2.3.2 for one coefficient, the remasking and decoding as described in Sect. 2.3.2 for four coefficients (that encode one bit), the masked χ-step of KECCAK for five bytes, point-wise multiplication and addition for two coefficients, two bits for the sampler, and 12 bytes for the comparison. To reduce the number of measured sample points per trace, we split the decoding into one measurement of the modulus transformation (Algorithm 1) and one measurement of the final operations as described in Algorithm 2. Figure 2.4 depicts the results for each module. The lower (resp. upper) curve shows the maximum absolute value of the first-order (resp. second-order) test as a function of the total number of measurements considered in the evaluation. It is noticeable that indeed no first-order leakage could be measured up to 100,000 traces. There is also no obvious increase of the t-values. Thus, the implementation showed first-order protection as expected. Additionally, the second-order evaluation shows leakage early on for every module and displays an upward trend with higher number of measurements. This is also expected given that we implemented first-order masking.

2.6 Results and Comparison

We evaluate the performance of our implementation using Keil μVision V5.17 and use `-O3` optimization for compiling. We took special care to prevent effects that the compiler optimization itself could induce side-channel leakage, e.g., by overwriting one shared value in a register with the second share. Cycle counts are measured using the on-board cycle count register (`DWT_CYCCNT`). To measure the dynamic memory consumption we used the callgraph feature of the Keil IDE. We present the cycle counts of our implementation in Table 2.1. The CCA2-secured encryption takes 4,176,684 cycles which translates to 25 ms when operating at a clock frequency of 168 MHz. The key generation takes 16 ms at 168 MHz.

Applying the CCA2-conversion to the decryption causes a much higher overhead due to the necessary re-encryption. In the unmasked case, it requires 27 times more cycles and in the masked case 46 times more cycles. Thus, the masked CCA2-decryption takes 25,334,493 cycles which is an overhead factor of 5.7 compared to the CCA2-secured decryption without masking. The overhead cost for the masking of the CCA2-secured decryption is mainly due to the high cost of the sampling. The sampling in turn heavily depends on the performance of the PRNG. A suitable replacement for SHAKE-128 would therefore drastically improve the

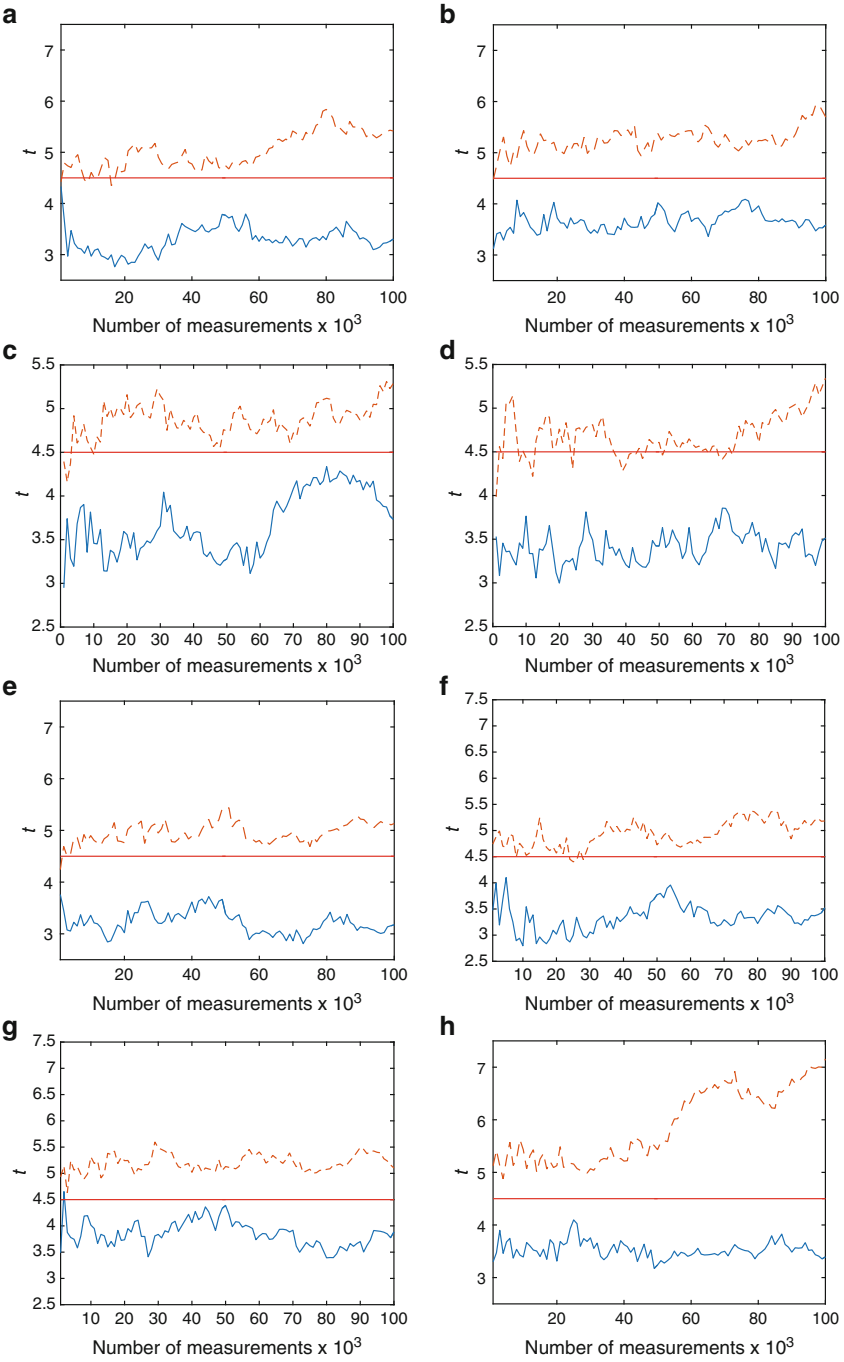

Fig. 2.4 Absolute maximum t-values for different modules of our masking scheme. The solid blue line marks the first-order t-values and the dashed red line marks the second-order t-values. (**a**) Butterfly. (**b**) Encryption. (**c**) Modulus transformation. (**d**) Final decoding operations. (**e**) Pointwise multiplication and addition. (**f**) Comparison. (**g**) Sampling. (**h**) Hash function

Table 2.1 Cycle counts of our implementation on an ARM Cortex-M4F

Operation	Cycle counts	
	Unmasked	Masked
Key generation ($\mathsf{RLWE.CPA}^{\mathrm{NTT}}_{\mathrm{gen}}$)	2,669,559	–
CCA2-secured encryption ($\mathsf{RLWE.CCA}^{\mathrm{NTT}}_{\mathrm{enc}}$)	4,176,684	–
CCA2-secured decryption ($\mathsf{RLWE.CCA}^{\mathrm{NTT}}_{\mathrm{dec}}$)	4,416,918	25,334,493
CPA-RLWE encryption ($\mathsf{RLWE.CPA}^{\mathrm{NTT}}_{\mathrm{enc}}$)	3,910,871	19,315,432
CPA-RLWE decryption ($\mathsf{RLWE.CPA}^{\mathrm{NTT}}_{\mathrm{dec}}$)	163,887	550,038
Shake-128	87,738	201,997
NTT	83,906	–
INTT	104,010	–
Uniform sampling (TRNG)	60,014	–
SampleNoisePoly (PRNG)	1,142,448	6,031,463
PRNG (64 bytes)	88,778	202,454

Cycle counts for sampling are given for a whole polynomial. Our parameters are $n = 1024$, $q = 12{,}289$, and $k = 8$

performance of the scheme. An insecure approach with an unmasked re-encryption would require around 2 million cycles only. However, as noted in Sect. 2.3.2 such an implementation would not provide sufficient protection against a side-channel adversary in a chosen-ciphertext scenario.

2.6.1 Comparison

Notice that the masked implementation in [43] is a hardware implementation and that [45] does not provide performance numbers. Thus we cannot directly compare our results to the existing work and decided to re-implement the previous proposals in combination with a CCA2-conversion to allow a fair comparison. Our results are given in Table 2.2. This also demonstrates the individual overhead of all schemes independent of the performance of the NTT. According to our findings, our CCA2-secured decryption needs one million cycles less than the masked decoder approach from [43] and 3.5 million cycles less than additively homomorphic masking [45]. It is also worth mentioning that encoding one message bit into four coefficients is much more complex when using the masked decoder approach as we no longer have $4^2 = 16$ possible combinations of values to match quadrants but $4^{2 \cdot 4} = 256$ combinations. Thus, for the evaluation of the masked decoding approach, we decode each coefficient separately and use masked majority voting to combine them. The additively homomorphic masking inherently increases the failure probability and may thus impact parameter choices and the acceptable noise levels.

Table 2.2 Cycle counts and dynamic memory consumption of our CCA2-secured decryption

Masking scheme	Cycle counts	Dynamic memory
Our scheme	25,334,493	25,696 bytes
Masked decoder [43]	26,250,420	25,696 bytes
Additively homomorphic masking [45]	28,899,058	29,984 bytes

2.7 Conclusion

In this chapter we presented the advanced Boolean and arithmetic techniques for masked decoding of cryptographic schemes in lattice-based encryption and KEMs. Besides our achievement of a reduced decryption failure probability, we also applied a CCA2-transform to the ring-LWE encryption scheme. This requires a masked sampling of the error polynomials. The implementation of our construction revealed that a side-channel and CCA2-secured implementation of ring-LWE comes with a significant overhead. We identified a target of further optimization within the masked implementation of the PRNG (SHAKE-128 in our case) for that further acceleration would result in a significantly increased overall performance.

References

1. Albrecht, M.R., Hanser, C., Höller, A., Pöppelmann, T., Virdia, F., Wallner, A.: Learning with errors on RSA co-processors. IACR Cryptol. ePrint Arch. **2018**, 425 (2018)
2. Alkim, E., Jakubeit, P., Schwabe, P.: A new hope on ARM Cortex-M. In: Carlet, C., Hasan, A., Saraswat, V. (eds). Security, Privacy, and Advanced Cryptography Engineering. Lecture Notes in Computer Science. Springer, Berlin (2016, to appear). Document ID: c7a82d41d39c535fd09ca1b032ebca1b. http://cryptojedi.org/papers/#newhopearm
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: Holz, T., Savage, S. (eds.). 25th USENIX Security Symposium, USENIX Security, 16, Austin, TX, pp. 327–343. USENIX Association, Berkeley, CA (2016)
4. Bai, S., Langlois, A., Lepoint, T., Stehlé, D., Steinfeld, R.: Improved security proofs in lattice-based cryptography: using the rényi divergence rather than the statistical distance. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology—ASIACRYPT 2015. Proceedings of the Part I: 21st International Conference on the Theory and Application of Cryptology and Information Security. Lecture Notes in Computer Science, vol. 9452, pp. 3–24. Springer, Berlin (2015)
5. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.X.: On the cost of lazy engineering for masked software implementations. In: Joye, M., Moradi, A. (eds.) 13th International Conference Smart Card Research and Advanced Applications (CARDIS 2014). Lecture Notes in Computer Science, vol. 8968, pp. 64–81. Springer, Berlin (2014)
6. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) Conference on the Theory and Application of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 263, pp. 311–323. Springer, Berlin (1986)
7. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 116–129. ACM, New York (2016)

8. Bernstein, D.J.: Cache-timing attacks on AES (2005)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G. Building power analysis resistant implementations of Keccak. In: Second SHA-3 Candidate Conference, vol. 142. (2010)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 7881, pp. 313–314. Springer, Berlin (2013)
11. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy, pp. 553–570. IEEE Computer Society, Silver Spring (2015)
12. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: take off the ring! Practical, quantum-secure key exchange from LWE. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1006–1018. ACM, New York (2016)
13. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D., CRYSTALS—Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy, p. 634. IEEE, Piscataway (2017)
14. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: Proceedings of the 12th USENIX Security Symposium. USENIX Association, Berkeley, CA (2003)
15. Buchmann, J.A., Göpfert, F., Güneysu, T., Oder, T., Pöppelmann, T.: High-performance and lightweight lattice-based public-key encryption. In: Chow, C., Saldamli, G. (eds.) Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security, pp. 2–9. ACM, New York (2016)
16. Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference (2013)
17. Coron, J.-S.: High-order conversion from Boolean to arithmetic masking. In: Fischer, W., Homma, N. (eds.) Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded Systems (CHES 2017). Lecture Notes in Computer Science, vol. 10529, pp. 93–114. Springer, Berlin (2017)
18. Coron, J-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to Boolean masking with logarithmic complexity. In: International Workshop on Fast Software Encryption, pp. 130–149. Springer, Berlin (2015)
19. Coron, J-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to Boolean masking with logarithmic complexity. In: Leander, G. (ed.) 22nd International Workshop on Fast Software Encryption (FSE 2015). Lecture Notes in Computer Science, vol. 9054, pp. 130–149. Springer, Berlin (2015)
20. De Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-LWE encryption. In: 2015 Design, Automation & Test in Europe Conference & Exhibition, p. 725. IEEE, Piscataway (2014)
21. Debraize, B.: Efficient and provably secure methods for switching from arithmetic to Boolean masking. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 107–121. Springer, Berlin (2012)
22. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In Wiener, M.J. (ed.) Proceedings of the 19th Annual International Conference on Advances in Cryptology (CRYPTO '99). Lecture Notes in Computer Science, vol. 1666, pp. 537–554. Springer, Berlin (1999)
23. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance validation. In: NIST Non-Invasive Attack Testing Workshop (2011)
24. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: Prouff, E., Schaumont, P. (eds). In: Proceedings of the 14th International Workshop on Cryptographic Hardware and

Embedded Systems (CHES 2012). Lecture Notes in Computer Science, vol. 7428. Springer, Berlin (2012), pp. 512–529. Springer, Berlin (2012)
25. Goubin, L.: A sound method for switching between Boolean and arithmetic masking. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001). Lecture Notes in Computer Science, vol. 2162, pp. 3–15. Springer, Berlin (2001)
26. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds). In: Proceedings of the 14th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2012). Lecture Notes in Computer Science, vol. 7428. Springer, Berlin (2012), pp. 530–547. Springer, Berlin (2012)
27. Howe, J., Oder, T., Krausz, M., Güneysu, T.: Standard lattice-based key encapsulation on embedded devices. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 372–393 (2018)
28. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Proceedings of the 23rd Annual International Conference on Advances in Cryptology (CRYPTO 2003), pp. 463–481. Springer, Berlin (2003)
29. Kannwischer, M.J., Rijneveld, J., Schwabe, P.: Faster multiplication in $\mathbb{F}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates. IACR Cryptol. ePrint Arch. **2018**, 1018 (2018)
30. Karmakar, A. Mera, J.M.B., Roy, S.S., Verbauwhede, I.: Saber on ARM CCA-secure module lattice-based key encapsulation on ARM. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 243–266 (2018)
31. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptograph. Eng. **1**(1), 5–27 (2011)
32. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias,A. (ed.) Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011. Lecture Notes in Computer Science, vol. 6558, pp. 319–339. Springer, Berlin (2011)
33. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010. Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer, Berlin (2010). Presentation slides: http://crypto.rd.francetelecom.com/events/eurocrypt2010/talks/slides-ideal-lwe.pdf
34. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Presentation of Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010. Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer, Berlin (2010). Presentation slides: http://crypto.rd.francetelecom.com/events/eurocrypt2010/talks/slides-ideal-lwe.pdf given by Chris Peikert at Eurocrypt'10 (2010). See http://www.cc.gatech.edu/~cpeikert/pubs/slides-ideal-lwe.pdf
35. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. IACR Cryptol. ePrint Arch. **2012**, 230 (2012). Full version of Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010. Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer, Berlin (2010). Presentation slides: http://crypto.rd.francetelecom.com/events/eurocrypt2010/talks/slides-ideal-lwe.pdf
36. National Institute of Standards and Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. Accessed 14 November 2018

37. Peikert, C.: Lattice cryptography for the Internet. In: Mosca, M. (ed) In: Proceedings of the 6th International Workshop on Post-Quantum Cryptography (PQCrypto 2014). Lecture Notes in Computer Science, vol. 8772, pp. 197–219. Springer, Berlin (2014)
38. Pöppelmann, T., Güneysu, T.: Towards practical lattice-based public-key encryption on reconfigurable hardware. In: Lange, T., Lauter, K.E.., Lisonek, P. (eds) 20th International Conference on Selected Areas in Cryptography (SAC 2013). Lecture Notes in Computer Science, vol. 8282, pp. 68–85. Springer, Berlin (2013)
39. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina, L., Robshaw, M., (eds.) In: Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2014). Lecture Notes in Computer Science, vol. 8731, pp. 353–370. Springer, Berlin (2014)
40. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds) Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded Systems (CHES 2017). Lecture Notes in Computer Science, vol. 10529, pp. 513–533. Springer, Berlin (2017)
41. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, pp. 84–93. ACM, New York (2005)
43. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-LWE implementation. In: Güneysu, T., Handschuh, H. (eds.) Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2015). Lecture Notes in Computer Science, vol. 9293, pp. 683–702. Springer, Berlin (2015)
44. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-LWE implementation. IACR Cryptol. ePrint Arch. **2015**, 724 (2015)
45. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively homomorphic ring-LWE masking. In: Takagi, T. (ed.) Proceedings of the 7th International Workshop on Post-Quantum Cryptography (PQCrypto 2016). Lecture Notes in Computer Science, vol. 9606, pp. 233–244. Springer, Berlin (2016)
46. Reparaz, O., Roy, S.S., de Clercq, R., Vercauteren, F., Verbauwhede, I.: Masking ring-LWE. J. Cryptograph. Eng. **6**(2), 139–153 (2016)
47. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact ring-LWE cryptoprocessor. In: Batina, L., Robshaw, M. (eds) Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2014). Lecture Notes in Computer Science, vol. 8731, pp. 371–391. Springer, Berlin (2014)
48. Saarinen, M.-J.O.: tiny_sha3 (2011). https://github.com/mjosaarinen/tiny_sha3
49. Saarinen, M.-J.O.: Arithmetic coding and blinding countermeasures for ring-LWE. IACR Cryptol. ePrint Arch. **2016**, 276 (2016)
50. Saarinen, M.-J.O., Bhattacharya, S., Garcia-Morchon, O., Rietman, R., Tolhuizen, L., Zhang, Z.: Shorter messages and faster post-quantum encryption with Round5 on Cortex M. IACR Cryptol. ePrint Arch. **2018**, 723 (2018)
51. Schneider, T., Moradi, A.: Leakage assessment methodology—a clear roadmap for side-channel evaluations. In: Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2015), pp. 495–513. Springer, Berlin (2015)
52. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE Computer Society, Silver Spring (1994)
53. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. In: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2010), pp. 112–129 (2010)

54. Targhi, E.E., Unruh, D.: Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In Hirt, M., Smith, A.D. (eds.) Part II: Proceedings of the 14th International Conference on Theory of Cryptography (TCC 2016-B). Lecture Notes in Computer Science, vol. 9986, pp. 192–216. Springer, Berlin (2016)
55. Trichina, E.: Combinational logic design for AES subbyte transformation on masked data. IACR Cryptol. ePrint Arch. **2003**, 236 (2003)
56. Vadnala, P.K., Großschädl, J.: Faster mask conversion with lookup tables. In: Mangard, S., Poschmann, A.Y. (eds.) 6th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2015). Lecture Notes in Computer Science, vol. 9064, pp. 207–221. Springer, Berlin (2015)
57. Von Zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, New York (2003)