Practical CCA2-Secure and Masked Ring-LWE Implementation

Tobias Oder[1], Tobias Schneider[2⋆], Thomas Pöppelmann[3], and Tim Güneysu[14]

[1]Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany
{tobias.oder,tim.gueneysu}@rub.de
[2]ICTEAM/ELEN/Crypto Group, Université Catholique de Louvain, Belgium
tobias.schneider@uclouvain.be
[3]Infineon Technologies AG, Germany
thomas.poeppelmann@infineon.com
[4]DFKI, Germany

Abstract. During the last years public-key encryption schemes based on the hardness of ring-LWE have gained significant popularity. For real-world security applications assuming strong adversary models, a number of practical issues still need to be addressed. In this work we thus present an instance of ring-LWE encryption that is protected against active attacks (i.e., adaptive chosen-ciphertext attacks) and equipped with countermeasures against side-channel analysis. Our solution is based on a post-quantum variant of the Fujisaki-Okamoto (FO) transform combined with provably secure, first-order masking. To protect the key and message during decryption, we developed a masked binomial sampler that secures the re-encryption process required by FO. Our work shows that CCA2-secured RLWE-based encryption can be achieved with reasonable performance on constrained devices but also stresses that the required transformation and handling of decryption errors implies a performance overhead that has been overlooked by the community so far. With parameters providing 233 bits of quantum security, our implementation requires 4,176,684 cycles for encryption and 25,640,380 cycles for decryption with masking and hiding countermeasures on a Cortex-M4F. The first-order security of our masked implementation is also practically verified using the non-specific t-test evaluation methodology.

1 Introduction

Public-key encryption (PKE) is a fundamental asymmetric cryptographic primitive and plays an extremely important role in numerous applications and security protocols, such as key-transport or email encryption. To date, most applications deploy RSA- and ECC-based schemes that are known to be broken by powerful quantum computers running Shor's polynomial-time algorithm [69] on a sufficiently large number of qubits. Given that such large-scale quantum computers

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

will exist in the future, the effects would be devastating as it would jeopardize the security of RSA or ECC protected ciphertexts exchanged today in case they are stored and decrypted in the future by a malicious entity[1]. Concerns over quantum computers have recently been fueled by an announcement of NIST to start the standardization process for post-quantum cryptography [19,51] and by the statement of NSA's Information Assurance Directorate (IAD) to "initiate a transition to quantum resistant algorithms in the not too distant future" for Suite B cryptography [50].

Possible candidates to replace RSA and ECC-based public-key encryption are cryptosystems based on the hardness of certain lattice problems – a very prominent example is NTRUEncrypt, proposed by Hoffstein, Pipher, and Silverman [39] almost two decades ago. More recently, cryptographic instances based on ideal lattices and the ring-learning with errors (ring-LWE) problem gained significant popularity in this field. This happened presumably due to their simplicity, high efficiency, and scalability (see [13, 14, 24, 65]), as well as because of theoretical foundations and security reductions (see [45, 53]). A practical advantage of ring-LWE-based encryption over NTRU is relatively easy constant-time implementation and fast key generation, which is useful when constructing schemes for ephemeral key exchange (e.g., NewHope [2] and BNCS [16]).

However, there are several challenges that have to be solved before ring-LWE-based encryption can be considered as a serious replacement of RSA or ECC for public-key encryption and (authenticated) key exchange. The most pressing and often overlooked issues are decryption errors (i.e., correctness), security against adaptive chosen-ciphertext attacks (CCA2)[2] and the protection against side-channel attacks. In this context, a basic semantically secure encryption scheme with parameters leading to a negligible amount of decryption errors is a requirement to achieve CCA2-security as discussed by Dwork, Naor, and Reingold [27] when applying CCA2-transformations. This issue was also shown by practical attacks on NTRU [40] or code-based encryption using QC-MDPC codes [37]. Moreover, CCA2-security is a condition for most real-world usage scenarios and has to be in place before side-channel protection can be considered. Otherwise, an attacker with physical access to a decryption oracle could simply create malformed ciphertexts to reveal a secret key, without the need to perform a side-channel attack at all[3]. The importance of CCA2-security is also reflected in the current NIST submission requirements for post-quantum public-key encryption and key-exchange [51] that explicitly ask for CCA2-security.

Contribution. In this work we address the aforementioned issues of ring-LWE PKE schemes that need to be considered before any wide-spread deploy-

[1] Note that this scenario is likely for PGP or S/MIME encrypted emails as users (or their providers) might keep them encrypted for a long time on a server over which they do not have control.

[2] Security against adaptive attacks is a stronger notion than security against non-adaptive chosen-ciphertext attacks where the adversary is constrained on the choice of the challenges, often referred to as CCA1 [9] or lunchtime attack.

[3] Note that a chosen-ciphertext attack on ring-LWE encryption [44, 45] is trivial as it is only secured against chosen plaintext attacks (CPA) [30].

ment of lattice-based cryptography can be initiated. We conservatively instantiate ring-LWE public-key encryption ($n = 1024$, $q = 12289$, and $\varsigma = 2$) for negligible decryption errors and implement the post-quantum variant of the Fujisaki-Okamoto [31] transformation by Targhi and Unruh [72]. Our main contribution is a novel, provably first-order secured masking scheme and its non-trivial integration into a CCA2 conversion. We point out that for full protection of the secret key and message in the probing model, a masked noise sampler is required for re-encryption and we provide the first design of corresponding protected binomial sampler. Our implementation and measurements were carried out on an ARM Cortex-M4F and we experimentally verified our masking scheme using the common non-specific t-test [32] methodology. In this setting, our implementation is the first instance for constrained devices that allows a fair comparison with established schemes like the Optimal Asymmetric Encryption Padding (OAEP) method with RSA or corresponding transformations for NTRU that can achieve CCA security. With masking and hiding countermeasures our code achieves 2,669,559 cycles for key generation, 4,176,684 cycles for encryption, and 25,640,380 cycles for decryption. The supposed security level against currently known quantum adversaries in the model of [2] is 233 bits. In comparison, our masking scheme thus outperforms previous masking approaches for ring-LWE by one million cycles.

Differences to previous version of this paper. In a previous version of this work [52] we presented an incomplete masking scheme in which we only masked the decryption of CPA-secure ring-LWE encryption ($\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}$). As described in Section 3.3 this is not sufficient to prevent first-order side-channel analysis. Furthermore we updated our proposal for the masking of $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}$ as our previous approach was leaking side-channel information in certain edge cases. We would like to thank the anonymous reviewers for pointing out these issues in the first version of this work.

2 Preliminaries

In this section we cover preliminaries on ring-LWE-based public-key encryption, discuss previous attempts to mask ring-LWE-based PKE schemes, and provide related work on protected NTRU implementations. 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 lower case letters. When we access a single bit of a bit vector, we use an index in square brackets to identify the respective bit.

2.1 Ring-LWE Encryption

The plain CPA-secured ring-LWE-based public-key encryption scheme we are using was previously proposed in [44,46,47]. Several variants of the scheme exist and the concrete instantiation we are using is defined as follows:

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

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 number theoretic transform (NTT)[4] in a way that has been previously described in [56, 65]. By $\tilde{\mathbf{a}}$ we denote that a polynomial $\mathbf{a}$ is in the NTT domain and 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 $(\mathbf{p} = \mathbf{r}_1 - \mathbf{a}\mathbf{r}_2, \mathbf{a})$ 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 [45]. In earlier works [16, 33] RLWE.CPA, or derived key exchange schemes, were 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 [2, 17], a fixed distribution [15], a binary distribution [18], or a uniform distribution [5, 36]. We define $\mathsf{SampleNoisePoly}()$ 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[5]. 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 ring-LWE. A uniformly random polynomial is sampled by $\mathsf{SampleUniformPoly}()$ 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 [2] 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

[4] The NTT basically allows to efficiently compute a polynomial multiplication $\mathbf{a} \cdot \mathbf{b}$ as $\mathbf{a} \cdot \mathbf{b} = \mathrm{INTT}(\mathrm{NTT}(\mathbf{a}) \circ \mathrm{NTT}(\mathbf{b}))$.

[5] In [2] the definition of the binomial distribution contains a typo in which the sum goes from zero to k.

(see [55, 67]). 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 $\mathsf{RLWE.CPA}$ 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 [56] (as mentioned earlier). The encoding function used in $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}$ is defined as $\mathsf{Encode}(m \in \{0,1\}^{n/4}) = \sum_{i=0}^{n-1} m[\lfloor i/4 \rfloor] \cdot q \cdot x^i$ (where $m[i]$ denotes the i-th bit of m). The decoding function used in $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}$ 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 Related Work on Masked Ring-LWE

Masking schemes for the ring-LWE encryption scheme have already been investigated by Reparaz, Roy, Vercauteren, and Verbauwhede in [62, 63]. The main idea of [62, 63] is to split the secret key $\mathbf{r}_2$ into two shares, compute the multiplication $\mathbf{r}_2 \cdot \mathbf{c}_1$ separately on both shares and add $\mathbf{c}_2$ to one of the shares. The authors construct a masked decoder that takes both shares as input and checks whether certain pre-defined rules are satisfied or not. For half of all inputs no rule applies and the value cannot be decoded immediately. This is solved by adding a certain $\delta \in [0, q-1]$ to the shares and restarting the decoding process up to 16 times. However, this process increases the decryption time and also the decryption error probability is increased by 19%, which has to be compensated by selecting lower noise sizes and thus leads to lower security.

In follow-up work Reparaz, de Clercq, Roy, Vercauteren, and Verbauwhede [60] propose a different masking scheme. The authors exploit that the ring-LWE decryption is *almost* additively homomorphic. Instead of dividing the secret key into two shares, they split the ciphertext into two shares and compute $\mathsf{decrypt}(\mathbf{c}'_1 + \mathbf{c}''_1, \mathbf{c}'_2 + \mathbf{c}''_2)$ to receive $(m' \oplus m'')$ as output. Note that this procedure includes an additional encryption of m'' during the decryption. Unfortunately, the addition of two ciphertexts implies that also the including error vectors are added and this again raises the decryption error probability of the scheme and lowers performance.

In both masking schemes, the decrypted message m is split into two parts, $m' = (m \oplus m'')$ and m''. In Appendix B of [64] the authors state that they were able to simulate a differential power analysis (DPA) attack targeting the pre-decoded value $\mathbf{z}$. As the output of the decryption is $m = \mathsf{LWEDecode}(\mathbf{z})$, we expect a DPA attack on m to be feasible as well. Thus, the shares of m must not

be combined on the device that performs the decryption. Note, that it might be a possibility to transmit the message already in two shares. However, this would mean that the protocol would have to be changed for this purpose; currently our approach could be easily adapted to a large number of lattice-based schemes. Additionally, the message space might not be sufficient. Another more severe issue is that the simulated attack on $\mathbf{z}$ from [64] requires an attacker to be able to choose arbitrary ciphertexts. Such an attacker is even able to find the secret key without DPA as ring-LWE itself does not provide CCA2-security (see [30]) but only security against chosen plaintext attacks (CPA). Thus, we draw two conclusions for the implementation of practically secured ring-LWE encryption:

- Assuming a CPA-only attacker, the DPA attack on ring-LWE without masked decoding is not feasible and thus no masked decoder is required.
- Assuming a CCA2 attacker, a CCA2-conversion has to be applied to ring-LWE. Otherwise, an attacker would be able to break the system without performing a DPA and thus rendering any side-channel countermeasures useless. The message m must not be stored unmasked in this setting.

As a consequence, the masking schemes described in [60, 62, 63] are less suitable for use in most practical settings.

2.3 Related Work on NTRU

In this section we review works on implementation attacks on NTRU. This is relevant as NTRU and ring-LWE have a similar structure (especially from the perspective of an implementer). Thus, (older) works on protecting NTRU are a natural reference for countermeasure to protect ideal (or even standard) lattice-based scheme that should not get overlooked.

In [4] a hardware implementation of NTRU and a first study regarding DPA attacks is provided. The attack allows recovering secret coefficients one-by-one using a Hamming distance model and Pearson's correlation coefficient. In [75] a correlation power analysis of an NTRU implementation equipped with the blinding countermeasures proposed in [49] is attacked. These countermeasures are addition of a random integer before convolution that can easily be removed, blinding using a random value, and randomization of the order of which coefficients are processed. As additional countermeasures in [75] random delays are proposed, masking, as well as dummy operations. A first order collision attack on NTRU is given in [77] and as countermeasure, besides random delays, a mathematical randomization is proposed where two inputs $\mathbf{a}$ and $\mathbf{b}$ to a convolution are randomly rotated as $\mathbf{a}' = \mathbf{a} \cdot x^i$ and $\mathbf{b}' = \mathbf{b} \cdot x^{n-i}$ for a random i so that the result $\mathbf{a}' \cdot \mathbf{b}' = \mathbf{a} \cdot x^i \cdot \mathbf{b} \cdot x^{n-i} = \mathbf{ab}$. The same countermeasure has recently also been proposed by Saarinen in [67] for lattice-based signatures with the observation that the shifting can be integrated into the NTT. Additionally, Saarinen proposes the multiplication with random constants that could also be integrated into the NTT computation. Timing attacks on NTRU have been investigated in [70]. Fault attacks are given in [42] and countermeasures against fault attacks are given in [43], mainly using spatial and temporal duplication.

Table 1: Security levels and failure probability of previously proposed ring-LWE-based public-key encryption or key-exchange schemes. The security level was computed based on the model in [2]. Note that C-Sec = classical bit-level security, Q-Sec = known quantum bit-level security, and that ς is the standard deviation of the ring-LWE noise distribution.

Set	Parameter (n, q, ς)	C-Sec	Q-Sec	Failure
RLWE.CPA [33]	$(256, 7681, \approx 4.5)$	64	58	$\approx 2^{-11}$
RLWE.CPA [33]	$(512, 12289, \approx 4.9)$	144	131	$\approx 2^{-10}$
BCNS [16]	$(1024, 2^{32} - 1, \approx 3.2)$	86	78	$2^{-2^{17}}$
NewHope [2]	$(1024, 12289, \approx 2.8)$	282	256	2^{-60}
This work	$(1024, 12289, 2)$	257	233	2^{-216}

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 [31, 72] (FO) transformation and our masking scheme.

3.1 Parameter Selection

To be able to use the RLWE.CPA scheme in the well-known hybrid setting a message space of 256-bit symmetric key is sufficient to account for quantum acceleration of brute-force attacks [35]. Additionally, to achieve CCA2-security using the FO transformation, a negligible error probability is required. As a consequence, previously proposed parameter sets, like the one used by the NewHope scheme [2] or RLWE.CPA scheme [33, 44], are not suitable. However, with $n = 1024$ we have four coefficients to encode one bit of a 256-bit message (similar as in NewHopeSimple [1]) and can thus tolerate noise levels $4 \cdot \frac{q}{4} = q$. To obtain a decryption error probability lower than 2^{-128} for the basic CPA-secured scheme we decreased the parameter of the binomial distribution used in [2] to $k = 8$ (contrary to $k = 16$ in NewHope). We therefore reach a noise level with standard deviation of $\varsigma = \sqrt{k/2} = \sqrt{8/2} = 2$. Even though a slightly larger value of k would also be possible, we opted for $k = 8$ as it gives a large enough margin on the error, simplifies and speeds-up sampling, and still leaves some room for more aggressive ciphertext compression (which is out of the scope of this work). A comparison of our final parameter set with previous proposals is given in Table 1 (see Table 5 and Appendix A.1 for more details and the parameters used for the scripts). Going from NewHope's 256-bits quantum security and 2^{-60} failure for $k = 16$ to 233 bits of quantum security and 2^{-216} failure probability for $k = 8$ seems like a reasonable trade-off. Note that the bit security level compared to NewHope is only slightly smaller and still more than 128-bits

of security against a known-quantum adversary[6]. Note that this level of security is still superior to BCNS or previously proposed RLWE.CPA parameters. It is also worth mentioning that the security estimation in [2] uses several worst-case estimations/simplifications so that the concrete security of the instances might be higher (i.e., there is currently no known algorithm that breaks, e.g., BCNS with 2^{86} steps).

3.2 CCA2 Conversion for RLWE.CPA

In this work we use the Fujisaki-Okamoto [31] transformation to enable a semantically secured encryption with respect to adaptive chosen ciphertext attack (CCA2). For this transformation, Peikert came to the conclusion [53] 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 [72]. 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 $\mathsf{RLWE.CCA}^{\mathsf{NTT}}_{\mathsf{enc}}$ and $\mathsf{RLWE.CCA}^{\mathsf{NTT}}_{\mathsf{dec}}$ as follows:

- $\mathsf{RLWE.CCA}^{\mathsf{NTT}}_{\mathsf{enc}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, m_{cca} \in \{0,1\}^L)$:
  Let $(\tilde{\mathbf{c}_1}, \mathbf{c}_2) = \mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \nu; H(\nu||m_{cca}))$ where $\nu \in \{0,1\}^L$ is a nonce and $H(\nu||m_{cca})$ seeds the PRNG of $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}$. Compute $c_3 = G(\nu) \oplus m_{cca}$ as well as $c_4 = H'(\nu)$ and output $(\tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)$.
- $\mathsf{RLWE.CCA}^{\mathsf{NTT}}_{\mathsf{dec}}(\tilde{\mathbf{r}}_2, \tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)$:
  Compute $\nu' = m_{cpa} = \mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}(\tilde{\mathbf{r}}_2, \tilde{\mathbf{c}}_1, \mathbf{c}_2)$, $m_{cca} = G(\nu') \oplus c_3$, $(\tilde{\mathbf{c}}_1^*, \mathbf{c}_2^*) = \mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \nu'; H(\nu'||m_{cca}))$, 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 $|(\tilde{\mathbf{a}}, \tilde{\mathbf{p}})| = 2n\lceil \log_2(q) \rceil = 2 \cdot 1024 \cdot 14 = 28672$ bits (3584 bytes) and a theoretical ciphertext size of $|(\tilde{\mathbf{c}}_1, \mathbf{c}_2, c_3, c_4)| = 2n\lceil \log_2(q) \rceil + 2l = 29184$ bits (3648 bytes). The secret key is $|\tilde{\mathbf{r}}_2| = n\lceil \log_2(q) \rceil = 14336$ bits (1792 bytes).

3.3 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 Figure 1 in bold notation, these modules are $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}$, G, H, H', and $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}$, and the two comparisons. Note that it is not sufficient to only protect $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{dec}}$, because

[6] In this context this means an adversary that uses quantum algorithms which are available today to accelerate cryptanalysis [1].

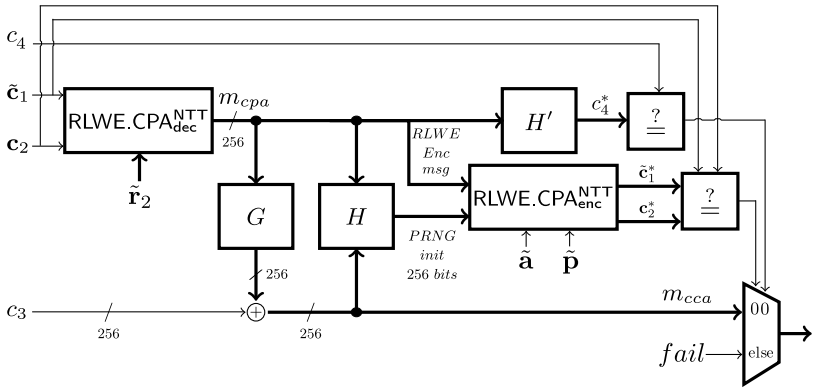


Fig. 1: CCA2-secured decryption.

in a chosen-ciphertext setting an adversary can target the unmasked output of RLWE.CPA$^{\mathsf{NTT}}_{\mathsf{dec}}$ (see Appendix B of [64]) 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 RLWE.CPA$^{\mathsf{NTT}}_{\mathsf{dec}}$ 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.

In the following, we analyze the first-order security of each module separately in the common probing model [41]. 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 [8].

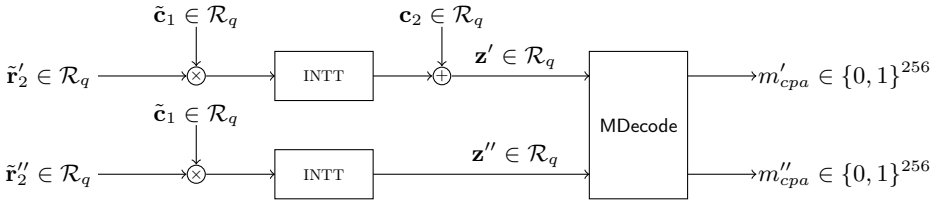


Fig. 2: Proposed masking scheme for ring-LWE decryption.

Ring-LWE Decryption. As mentioned in Section 2.2, the masking schemes of the ring-LWE decryption from works like [62,63] and [60] 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 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 [62, 63]. 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 [62,63], 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 [60].

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 [62, 63] 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 [21,22,25,34,74] 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 $y_1 + y_2 \bmod 2^{15} = x$ given that 2^{15} is the second-next-larger power of two for $q = 12289$. 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

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

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 $z_1 \leftarrow$

$y_1 - q \bmod 2^{bits}$. By doing this, we create the following relation for the most significant bit of $z_1 + y_2 \bmod 2^{bits}$

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

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 [25], 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

$$\begin{aligned} y_1 - (k_1 \oplus k_2)q = \; & y1 - k_1q - k_2q + 2k_1k_2q \\ = \; & y_1 - k_1q - k_2q + 2(k'_1 + k''_1)(k'_2 + k''_2)q \\ = \; & y_1 - k_1q - k_2q + 2k'_1k'_2q + 2k'_1k''_2q + 2k''_1k'_2q + 2k''_1k''_2q. \end{aligned}$$

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.

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: $y_1 = (((((((r + y_1) - k_1q) - k_2q) + 2k'_1k'_2q) + 2k'_1k''_2q) + 2k''_1k'_2q) + 2k''_1k''_2q)$
14: $y_2 = y_2 - r$

Although the output (y_1, y_2) of TransformPower2 fulfils 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 Figure 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)

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

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 combined 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 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 $q = 12289$. 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.

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: $[a_1, a_2] \leftarrow \texttt{TransformPower2}(a_1, a_2, 16)$
6: $[b_1, b_2] \leftarrow \texttt{TransformPower2}(b_1, b_2, 16)$
7: $[c_1, c_2] \leftarrow \texttt{TransformPower2}(c_1, c_2, 16)$
8: $[d_1, d_2] \leftarrow \texttt{TransformPower2}(d_1, d_2, 16)$
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: $[e_1, e_2] \leftarrow \texttt{A2B}(e_1, e_2)$
13: $m_1 = \texttt{MSB}(e_1)$
14: $m_2 = \texttt{MSB}(e_2)$

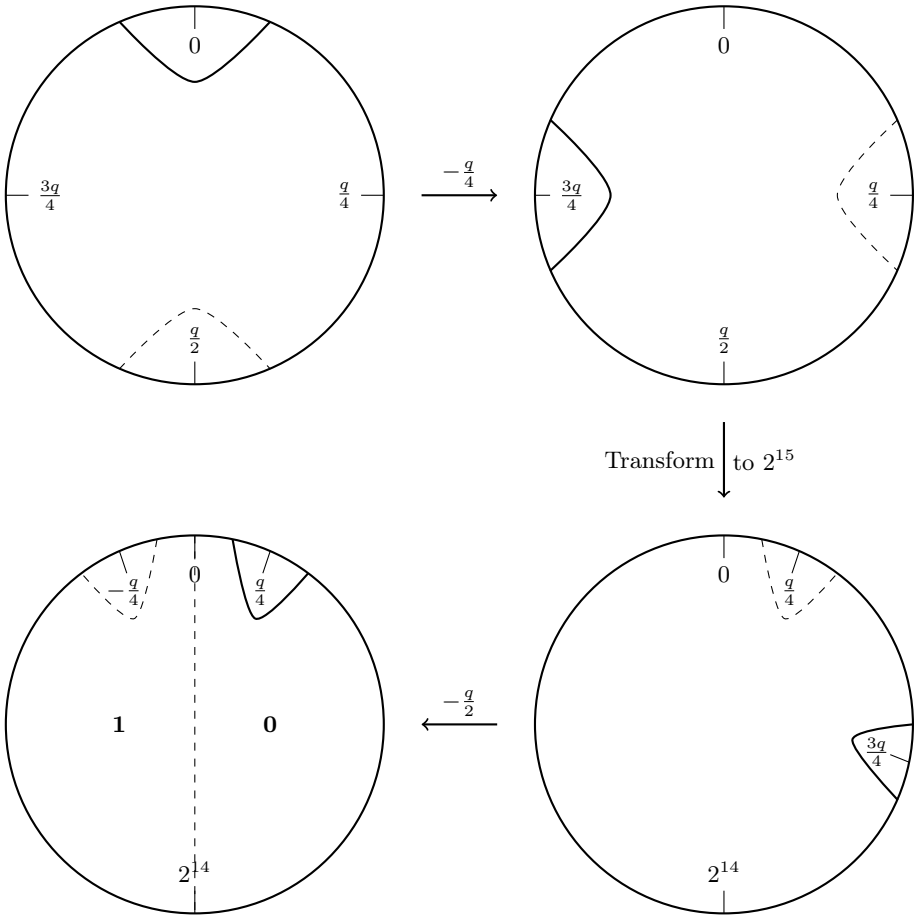


Fig. 3: First three steps when decoding one coefficient.

Security Analysis We analyze the security of Algorithm 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 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 $(x_1 - y_1) + x_2 = x - y_1$.
- *Lines 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 [25] and refer the interested reader to their proof of security.
- *Line 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 $x = y_1 + y_2 \bmod 2^{bits}$ are both uniformly distributed in $\{0,1\}^{bits}$.

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

Lemma 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.
– *Line 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`.
– *Line 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 [11]. 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 $\mathsf{RLWE.CPA}^{\mathsf{NTT}}_{\mathsf{enc}}$ (i.e. the re-encryption in $\mathsf{RLWE.CCA}^{\mathsf{NTT}}_{\mathsf{dec}}$), 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 [53] 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.

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

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 $(\mathbf{m'_{cpa,1}} + \mathbf{m'_{cpa,2}}) \circ (\mathbf{m''_{cpa,1}} + \mathbf{m''_{cpa,2}})$ instead of m'_{cpa} `AND` m''_{cpa}. To obtain the correct result, we compute:

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

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` [73]. 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 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* $\mathbf{m'_{cpa,1}}, \mathbf{m''_{cpa,1}} \in \mathcal{R}_q$ *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 , $\mathbf{e}_3$.

Algorithm 3 Masked Ring-LWE Encryption

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

- *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}.
- *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.
- *Lines 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:

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

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.

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 $(out_1 + out_2)$ mod q binomial distributed

1: $i \leftarrow 0$
2: $out_1 \leftarrow 0$
3: $out_2 \leftarrow 0$
4: **for** $i < k$ **do**
5: $\quad out_1 = out_1 + (\alpha_1[i] - \beta_1[i])$
6: $\quad out_2 = out_2 + (\alpha_2[i] - \beta_2[i])$
7: $\quad X \xleftarrow{\$} [0, q-1], Y \xleftarrow{\$} [0, q-1], Z \xleftarrow{\$} [0, q-1]$
8: $\quad \alpha_1'' = \alpha_1 - X$
9: $\quad \alpha_2'' = \alpha_2 - Y$
10: $\quad out_1 = out_1 - 2((((Z + XY) + X\alpha_2'') + \alpha_1''Y) + \alpha_1''\alpha_2'')$
11: $\quad \beta_1'' = \beta_1 - X$
12: $\quad \beta_2'' = \beta_2 - Y$
13: $\quad out_2 = out_2 + 2((((Z + XY) + X\beta_2'') + \beta_1''Y) + \beta_1''\beta_2'')$
14: $\quad i \leftarrow i + 1$
15: **end for**

Lemma 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.
- *Line 8-13*: The proof works analogous to the proof for Lines 5-9 of Lemma 3.

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.

3.4 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

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: $eq \leftarrow (eq == 0)$

approach is easy to verify as

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

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 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 $\tilde{\mathbf{c}}_1^*$.
- *Lines 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.

3.5 Hiding

To increase the level of noise and make higher-order attacks harder, we do not only rely on masking to thwart side-channel analysis but also include hiding schemes. We therefore applied the aforementioned blinding technique from [67] to our implementation by multiplying the coefficients of $\mathbf{c}_1$ by a random value $a \in [0, q-1]$ and the coefficients of $\mathbf{r}_2$ by a different random value $b \in [0, q-1]$. The coefficients of $\mathbf{c}_2$ are multiplied by $ab \mod q$ as they get added to the product $(a \cdot \mathbf{c}_1) \cdot (b \cdot \mathbf{r}_2)$. The mask is then removed by multiplying all coefficients by $(ab)^{-1} \mod q$. Due to the linearity of the NTT it is possible to remove the mask after the result has been transformed back to the time domain. To introduce even more noise we used shuffling to execute linear operations during the decryption in a randomized order. To achieve this we shuffled the list of coefficients by using the Fisher-Yates algorithm [29]. A similar countermeasure has been implemented by Pessl [54] to avoid cache-timing attacks. For every run of the decryption, the list of coefficients gets shuffled again.

3.6 Fault Resistance

Fault injection is an additional physical threat for embedded systems. Previous publications have analyzed the vulnerability of lattice-based signature schemes against fault attacks [12, 28] and found several attacks. In the following, we present the – to our knowledge – first vulnerability analysis of ring-LWE.

In our analysis, we assume that the adversary targets the secret key $\mathbf{r}_2$ during the CCA2-secured decryption. Without CCA2-security, a fault attack is not necessary to recover the secret key as described before. Given that the CCA2-conversion includes a validity check at the end, it inherently includes resistance

against certain faults. In particular, any fault injected in the ring-LWE decryption that changes the output of the ring-LWE decryption module will be detected by the re-encryption at the end[7]. Therefore, to perform any type of fault attack, it is required to inject another fault into the input of H (to change the seed) or the ring-LWE encryption with the goal of passing the validity check. Depending on the capabilities of the fault attacker, this approach can be very complex. Therefore, in most cases it is easier to directly inject the fault in the validity check itself, e.g., by skipping instructions.

Furthermore, due to the construction of our scheme the attacker does not have direct access to the output of the ring-LWE decryption module. Instead, the output is defined as

$$\begin{aligned} out &= c_3 \oplus G(m) \\ &= (M \oplus G(m)) \oplus G(m) = M \end{aligned}$$

where M is the message and m the output of the ring-LWE decryption. Assuming that the attacker knows M and $G(m)$, the faulty output is

$$out_F = c_3 \oplus G(m_F)$$

and therefore the only novel information the attack can access is $G(m_F)$. Based on the pre-image resistance of G, it is not easily possible to compute m_F from $G(m_F)$ for arbitrary m_F. This poses another difficulty for the fault attacker, as it is necessary to skip the computation of G to perform attacks that target the output of the decryption assuming m_F is uniformly distributed in $\{0,1\}^{n/4}$.

However, it is possible to overcome this limitation. A much simpler attack relies on the basic vulnerability of ring-LWE decryption to chosen ciphertexts. By skipping the validity check at the end, the attacker effectively removes the CCA2-security. Meaning an attacker can send chosen ciphertexts and receive the output

$$out = c_3 \oplus G(m_C)$$

where m_C denotes the output of the decryption for the chosen ciphertext. Even though, we noted above that G provides pre-image resistance, this does not apply when m_C has only a very limited value space. Then it is possible to compute $G(m_C)$ for all possible m_C and use *out* to check for the correct one. For our implementation, an attacker can target each coefficient of the secret key polynomial separately by choosing $\tilde{\mathbf{c}}_1$ as a polynomial with all coefficients but one set to zero. Therefore, an attacker needs to compute only q different values for m_C. Overall, this attack only requires the injection of one fault at the end to skip the validity check of the CCA2-conversion.

In conclusion, our implementation provides basic resistance against simple faults in the ring-LWE decryption. However, if the attacker can skip the validity

[7] There are no two distinct outputs of the decryption that can be valid at the same time

check, it becomes very easy to extract the secret key. Therefore, to increase the resistance against physical attacks, additional countermeasures need to be included to protect this final check. Furthermore, more sophisticated attacks, e.g., safe-error attacks [76], also pose a threat to the secret key and, therefore, would require more sophisticated fault countermeasures.

3.7 Higher-Order Masking

As mentioned before, it is not sufficient to analyze the security against d probes for each module separately to show the security of the full decryption. Nevertheless, we now briefly discuss the possible extension of our masked modules to higher-orders.

For the first part of $\mathsf{RLWE.CPA}_{\mathsf{dec}}^{\mathsf{NTT}}$, each share is processed separately and therefore extending the security to more probes is trivial. A designer only needs to increase the number of shares and process them similar to $\mathbf{z}''$. However, the addition of the shares in the second part requires special care, e.g., order of operation, to obtain higher-order security.

For G, H, and H' we refer to [11] for a discussion of higher-order resistance. We want to note that for this module the efficiency strongly depends on the chosen function. Keccak is efficient for first-order security. However, for higher orders a different function might be better suited. In the module $\mathsf{RLWE.CPA}_{\mathsf{enc}}^{\mathsf{NTT}}$most operations are linear and, therefore, can be trivially extended to higher orders. Only the masked AND needs to be extended to higher orders as described by Ishai, Sahai, and Wagner [41]. For the masked sampler, the extension to higher orders would be quite expensive as it is already the largest part of the implementation. Therefore, it is reasonable to design a new sampler which is easy to mask. Maintaining an additional share means that we need extra temporary storage for one polynomial that stores the third share of the key (2048 bytes) and one additional Keccak state (200 bytes). Our target platform provides 192 kbytes of RAM and therefore we expect that a higher-order masking scheme still fits onto the microcontroller.

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. This also prevents the single-trace attack by Primas, Pessl, and Mangard [57] that exploits timing differences in the `DIV` instruction.

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 [66] 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 [2] 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 arithmetic-related SPA and remote timing attacks [57]. As a consequence, the implementation of the NTT and especially the three-instruction modular reduction from [23] 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 [7] using the FPU of the Cortex-M4F that takes 6 clock cycles and is timing-independent. De Clercq et al. [23] 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 [3]. By combining a Montgomery reduction with Barrett reduction, their NTT is considerably faster than the one from [23] and most importantly also has a constant execution time. We therefore embedded the NTT from [3] into our implementation.

For our implementation of the blinding countermeasure, we need to compute the inverse of the product of the blinding values $(ab)^{-1} \mod q$. To realize this inversion efficiently, we used an addition chain to compute $(ab)^{q-2} \mod q$ what equals $(ab)^{-1} \mod q$ according to Fermats little theorem. To shuffle the list of indices of the polynomials and therefore change the order of the computations according to the Fisher-Yates algorithm [29].

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 op-

erations that require both shares (i.e. Decode) hand-crafted assembly code is necessary.

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 [6]. 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 [20,32]. We performed the test at first and second order. For bivariate second-order evaluation, we relied on the optimal centered product [58,71] 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 Section 3.3. For first-order evaluation this is appropriate as noted in Section 3.3. 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 [68].

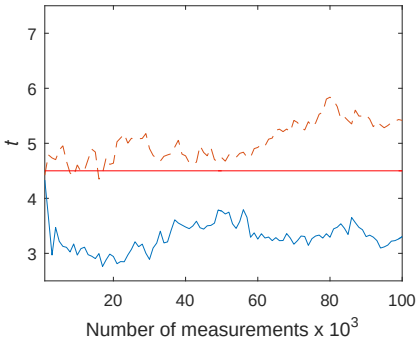
We measured the computation of the butterfly during the NTT for two coefficients, the addition of the two shares during the masked re-encryption as described in Section 3.3 for one coefficient, the remasking and decoding as described in Section 3.3 for four coefficients (that encode one bit), the masked

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 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.

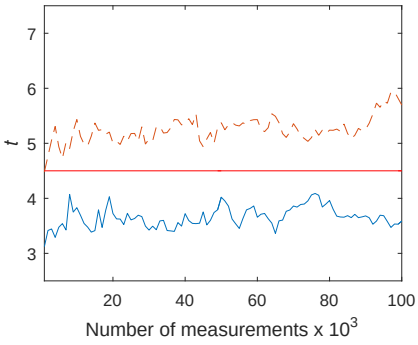
However, it should be noted that even though our evaluations clearly show second-order leakage, performing an actual second-order attack on the masked implementation might not be trivial. First, there is the aforementioned issue of the extreme high number of sample points which can make our naive combination approach unfeasible. Instead, more sophisticated point of interest detection mechanisms need to be utilized to reduce the number of considered sample pairs [26, 61] which further increases the complexity of the attack. A second aspect which is briefly mentioned in Section 3.5 is the mixture of masking with hiding countermeasures. Higher-order attacks are very sensitive to noise. Therefore, to increase the higher-order security it is advised to include one of the discussed hiding countermeasures. In one recent example, this increased the practical resistance more than implementing a higher-order masking scheme [48]. Furthermore, if only the start of the ring-LWE decryption is targeted, a designer can rely on the linear masking property to increase the number of shares significantly. However, including hiding countermeasures prevents us from evaluating only simplified versions of the modules, since the efficiency strongly increases with the number of coefficients. Therefore, we did not measure a masked design with hiding, since a thorough second-order evaluation would not be feasible (the first-order test would give similar results to Figure 4).

6 Results and Comparison

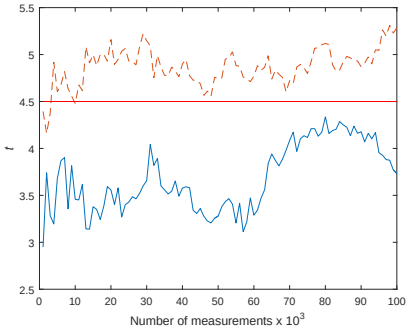
We evaluate the performance of our implementation using Keil μVision V5.17 and use `-O3` optimization for compiling. We took special care to prevent effects that the compiler optimization itself could induce side-channel leakage, e.g. by overwriting one shared value in a register with the second share. 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. The CCA2-secured encryption takes 4,176,684 cycles which translates to 25 milliseconds



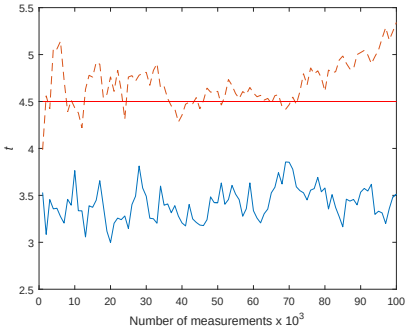
(a) Butterfly



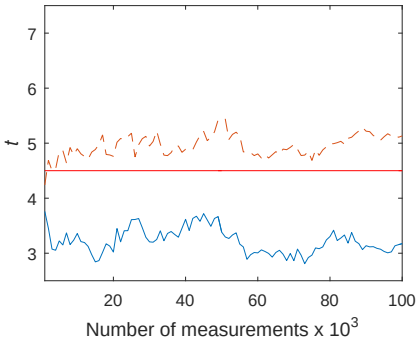
(b) Encryption



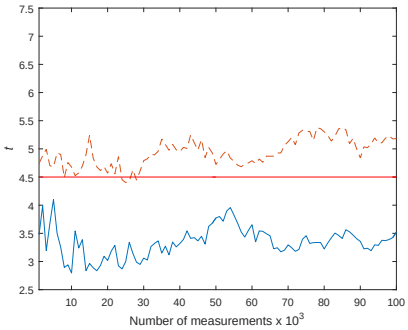
(c) Modulus Transformation



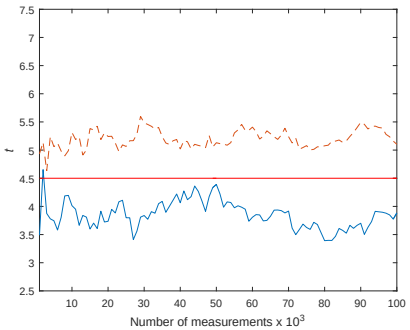
(d) Final decoding operations



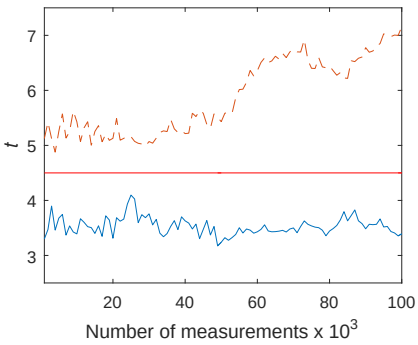
(e) Point-wise multiplication and addition



(f) Comparison



(g) Sampling



(h) Hash function

Fig. 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.

Table 2: Cycle counts of our implementation on an ARM Cortex-M4F. Cycle counts for sampling are given for a whole polynomial. Our parameters are n = 1024, q = 12289, and k = 8.

Operation	**Cycle Counts**	
	Unmasked	**Masked**
Key Generation ($\mathsf{RLWE.CPA}_{\mathsf{gen}}^{\mathsf{NTT}}$)	2,669,559	-
CCA2-secured Encryption ($\mathsf{RLWE.CCA}_{\mathsf{enc}}^{\mathsf{NTT}}$)	4,176,684	-
CCA2-secured Decryption ($\mathsf{RLWE.CCA}_{\mathsf{dec}}^{\mathsf{NTT}}$)	4,416,918	25,334,493
CPA-RLWE Encryption ($\mathsf{RLWE.CPA}_{\mathsf{enc}}^{\mathsf{NTT}}$)	3,910,871	19,315,432
CPA-RLWE Decryption ($\mathsf{RLWE.CPA}_{\mathsf{dec}}^{\mathsf{NTT}}$)	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

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 performance of the scheme. An insecure approach with an unmasked re-encryption would require around 2 million cycles only. However, as noted in Section 3.3 such an implementation would not provide sufficient protection against a side-channel adversary in a chosen-ciphertext scenario.

The results of combining our masking approach of the decryption with additional hiding countermeasures are given in Table 3. The overall running time for our protected decryption is 25,640,380 cycles which leads to 152 milliseconds runtime at 168 MHz. The secret key is one polynomial and therefore requires 2,048 bytes of memory. As the public key consists of two polynomials, it needs twice as much memory (4,096 bytes). The ciphertext consists of two polynomials $(\tilde{\mathbf{c}}_1, \mathbf{c}_2)$ and the bit strings c_3, c_4 of 256 bits each and therefore has a total size of 4,136 bytes.

Table 3: Cycle counts of our CCA2-secured decryption.

Countermeasure	Masking	
	Unmasked	**Masked**
No Hiding	4,416,918	25,334,493
Blinding/Shuffling	4,643,394	25,640,380

6.1 Comparison

Notice that the masked implementation in [63] is a hardware implementation and that [60] 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 Tabel 4. 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 [63] and 3.5 million cycles less than additively homomorphic masking [60]. 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 4: 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 [63]	26,250,420	25,696 bytes
Additively homomorphic masking [60]	28,899,058	29,984 bytes

7 Conclusion

In this work we presented a new instantiation of CPA-secured ring-LWE encryption with masked decoding that outperforms previous proposals at 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.

Acknowledgement

The research in this work was supported in part by the DFG Research Training Group GRK 1817/1 and by the European Unions Horizon 2020 program under project number 645622 PQCRYPTO, 644729 SAFEcrypto, and 724725 SWORD. We would also like to thank the anonymous reviewers for their very valuable and helpful feedback.

References

1. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Newhope without reconciliation. *IACR Cryptology ePrint Archive*, 2016:1157, 2016. 7, 8
2. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016. 2, 3, 4, 7, 8, 23, 36
3. Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. A new hope on ARM Cortex-M. In Claude Carlet, Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Advanced Cryptography Engineering*, Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2016 (to appear). Document ID: c7a82d41d39c535fd09ca1b032ebca1b, http://cryptojedi.org/papers/#newhopearm. 23
4. AC Atici, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on NTRU implementations for RFIDs: First results. In *The 4th Workshop on RFID Security, July 9th -11th, Budapest*, 2008. 6
5. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015. 4
6. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014. 24

7. Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1986. 23
8. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016. 9
9. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998. 2
10. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013. 15
11. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of Keccak. In *Second SHA-3 candidate conference*, volume 142. Citeseer, 2010. 15, 22
12. Nina Bindel, Johannes A. Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. *IACR Cryptology ePrint Archive*, 2016:415, 2016. to appear in FDTC'16. 20
13. Ahmad Boorghany and Rasool Jalili. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. *IACR Cryptology ePrint Archive*, 2014:78, 2014. 2
14. Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. *IACR Cryptology ePrint Archive*, 2014:514, 2014. 2
15. Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. *IACR Cryptology ePrint Archive*, 2016:659, 2016. to appear in CCS'16. 4
16. Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 553–570. IEEE Computer Society, 2015. 2, 4, 7, 36
17. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. *IACR Cryptology ePrint Archive*, 2017:634, 2017. 4
18. Johannes A. Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. High-performance and lightweight lattice-based public-key encryption. In Richard Chow and Gökay Saldamli, editors, *Proceedings of the 2nd ACM*

International Workshop on IoT Privacy, Trust, and Security, IoTPTS@AsiaCCS, Xi'an, China, May 30, 2016, pages 2–9. ACM, 2016. 4

19. Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. DRAFT NISTIR 8105, report on post-quantum cryptography, 2015. http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf. 2
20. Jeremy Cooper, Elke Demulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test Vector Leakage Assessment (TVLA) Methodology in Practice. International Cryptographic Module Conference, 2013. 24
21. Jean-Sébastien Coron. High-order conversion from boolean to arithmetic masking. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 93–114. Springer, 2017. 10
22. Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2015. 10
23. Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient software implementation of ring-LWE encryption. *IACR Cryptology ePrint Archive*, 2014:725, 2014. 23
24. Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient software implementation of ring-LWE encryption. In Wolfgang Nebel and David Atienza, editors, *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 339–344. ACM, 2015. 2
25. Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to boolean masking. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 107–121. Springer, 2012. 10, 11, 14
26. François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 34–50, 2015. 25
27. Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 342–360. Springer, 2004. 2
28. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop abort faults on lattice-based Fiat-Shamir & hash'n sign signatures. *IACR Cryptology ePrint Archive*, 2016:449, 2016. 20
29. Ronald Aylmer Fisher, Frank Yates, et al. *Statistical tables for biological, agricultural and medical research.* Edinburgh and London: Oliver & Boyd., 1957. 5th rev. ed. 20, 23
30. Scott R. Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016. 2, 6

31. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999. 3, 7, 8
32. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011. 3, 24
33. Norman Göttert, Thomas Feller, Michael Schneider, Johannes A. Buchmann, and Sorin A. Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In Prouff and Schaumont [59], pages 512–529. 4, 7, 36
34. Louis Goubin. A sound method for switching between boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001. 10
35. Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996. 7
36. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Prouff and Schaumont [59], pages 530–547. 4
37. Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 789–815, 2016. 2
38. Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRUEncrypt. IACR Cryptology ePrint Archive report 2015/708, 2015. http://eprint.iacr.org/2015/708. 36
39. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998. 2
40. Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 226–246. Springer, 2003. 2
41. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 463–481, 2003. 9, 22
42. Abdel Alim Kamal and Amr M. Youssef. Fault analysis of the NTRUEncrypt cryptosystem. *IEICE Transactions*, 94-A(4):1156–1158, 2011. 6

43. Abdel Alim Kamal and Amr M. Youssef. Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks. *J. Cryptographic Engineering*, 3(4):227–240, 2013. 6
44. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011. 2, 3, 7
45. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010. Presentation slides: http://crypto.rd.francetelecom.com/events/eurocrypt2010/talks/slides-ideal-lwe.pdf. 2, 4, 33
46. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings, 2010. Presentation of [45] given by Chris Peikert at Eurocrypt'10. See http://www.cc.gatech.edu/~cpeikert/pubs/slides-ideal-lwe.pdf. 3
47. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *IACR Cryptology ePrint Archive*, 2012:230, 2012. Full version of [45]. 3
48. Amir Moradi and Alexander Wild. Assessment of hiding the higher-order leakages in hardware - what are the achievements versus overheads? In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 453–474, 2015. 25
49. LEE Mun-Kyu, Jeong Eun Song, and HAN Dong-Guk. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 93(1):153–163, 2010. 6
50. National Security Agency. Commercial national security algorithm suite, 2015. https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm, Last Reviewed on August 19, 2015. 2
51. NIST. Request for comments on post-quantum cryptography requirements and evaluation criteria. *National Institute of Standards and Technology*, August 2016. See https://federalregister.gov/a/2016-18150. 2
52. Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Cryptology ePrint Archive*, 2016:1109, 2016. 3
53. Chris Peikert. Lattice cryptography for the Internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014. 2, 8, 16
54. Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, pages 153–170. Springer, 2016. 20
55. Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International*

Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014. 5

56. Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2013. 4, 5
57. Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017. 22, 23
58. Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009. 24
59. Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*. Springer, 2012. 32
60. Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-LWE masking. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2016. 5, 6, 10, 28
61. Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Selecting time samples for multivariate DPA attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 155–174, 2012. 25
62. Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-LWE. *J. Cryptographic Engineering*, 6(2):139–153, 2016. 5, 6, 10
63. Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-LWE implementation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 683–702. Springer, 2015. 5, 6, 10, 28
64. Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-LWE implementation. *IACR Cryptology ePrint Archive*, 2015:724, 2015. 5, 6, 9
65. Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-LWE cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 371–391. Springer, 2014. 2, 4
66. Markku-Juhani O. Saarinen. tiny_sha3. https://github.com/mjosaarinen/tiny_sha3, 2011. 23
67. Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for ring-LWE. *IACR Cryptology ePrint Archive*, 2016:276, 2016. 5, 6, 20

68. Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 495–513, 2015. 24
69. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994. 1
70. Joseph H. Silverman and William Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2007. 6
71. François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 112–129, 2010. 24
72. Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the fujisaki-okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216, 2016. 3, 7, 8
73. Elena Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptology ePrint Archive*, 2003:236, 2003. 16
74. Praveen Kumar Vadnala and Johann Großschädl. Faster mask conversion with lookup tables. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2015. 10
75. An Wang, Xuexin Zheng, and Zongyue Wang. Power analysis attacks and countermeasures on NTRU-based wireless body area networks. *TIIS*, 7(5):1094–1107, 2013. 6
76. Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000. 22
77. Xuexin Zheng, An Wang, and Wei Wei. First-order collision attack on protected NTRU cryptosystem. *Microprocessors and Microsystems - Embedded Hardware Design*, 37(6-7):601–609, 2013. 6

A Appendix

A.1 Security Estimation

In Table 5 we provide details security estimations based on the approach and using the script provided in [2]. For the evaluation of the error rate we also used the script from [2] and set the parameters $dim = 1024, q = 12289, k = 8, bound\_CC = 66500, t\_CC = 0.0055, tau = 17.8$. To evaluate the security level of our proposal we add the line

```
summarize_params(12289,1024,sqrt(8.0/2), 3.*12289.0/4, False)
```

to the original NewHope script[8].

Table 5: Security level of various parameters for ring-LWE encryption schemes

Attack	m	b	Known Classical	Known Quantum	Best Plausible
RLWE.CPA [33] $q = 7681$, $n = 256$, $\varsigma \approx 4.5160$					
Primal	347	222	64	58	46
Dual	369	222	64	58	46
RLWE.CPA [33] $q = 12289$, $n = 512$, $\varsigma \approx 4.8591$					
Primal	660	496	145	131	102
Dual	674	494	144	131	102
BCNS proposal [16]: $q = 2^{32} - 1$, $n = 1024$, $\varsigma = 3.192$					
Primal	1062	296	86	78	61
Dual	1055	296	86	78	61
NTRUEncrypt [38]: $q = 2^{12}$, $n = 743$, $\varsigma \approx \sqrt{2/3} \approx 0.8165$					
Primal	613	603	176	159	125
Dual	635	600	175	159	124
JarJar: $q = 12289$, $n = 512$, $\varsigma = \sqrt{12} \approx 3.4641$					
Primal	623	449	131	119	93
Dual	602	448	131	118	92
NewHope: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{8} \approx 2.8284$					
Primal	1100	967	282	256	200
Dual	1099	962	281	255	199
Our Work: $q = 12289$, $n = 1024$, $\varsigma = \sqrt{8/2} = 2$					
Primal	999	886	259	235	183
Dual	1048	881	257	233	182

[8] See PQsecurity.py in https://cryptojedi.org/crypto/data/newhope-20160815.tar.bz2