Higher-Order Masked Ciphertext Comparison for Lattice-Based Cryptography

Jan-Pieter D'Anvers[1], Daniel Heinz[2,3], Peter Pessl[3], Michiel Van Beirendonck[1] and Ingrid Verbauwhede[1]

[1] imec-COSIC KU Leuven, Kasteelpark Arenberg 10 - bus 2452, 3001 Leuven, Belgium
{firstname}.{lastname}@esat.kuleuven.be
[2] Research Institute CODE, Universität der Bundeswehr München, 85577 Neubiberg, Germany
daniel.heinz@unibw.de
[3] Infineon Technologies, Am Campeon 1-15, 85579 Neubiberg, Germany
peter.pessl@infineon.com

Abstract.
Checking the equality of two arrays is a crucial building block of the Fujisaki-Okamoto transformation, and as such it is used in several post-quantum key encapsulation mechanisms including Kyber and Saber. While this comparison operation is easy to perform in a black box setting, it is hard to efficiently protect against side-channel attacks. For instance, the hash-based method by Oder et al. is limited to first-order masking, a higher-order method by Bache et al. was shown to be flawed, and a very recent higher-order technique by Bos et al. suffers in runtime. In this paper, we first demonstrate that the hash-based approach, and likely many similar first-order techniques, succumb to a relatively simple side-channel collision attack. We can successfully recover a Kyber512 key using just 6000 traces. While this does not break the security claims, it does show the need for efficient higher-order methods. We then present a new higher-order masked comparison algorithm based on the (insecure) higher-order method of Bache et al. Our new method is 4.2x, resp. 7.5x, faster than the method of Bos et al. for a 2^{nd}, resp. 3^{rd}, -order masking on the ARM Cortex-M4, and unlike the method of Bache et al., the new technique takes ciphertext compression into account. We prove correctness, security, and masking security in detail and provide performance numbers for 2^{nd} and 3^{rd}-order implementations. Finally, we verify our the side-channel security of our implementation using the test vector leakage assessment (TVLA) methodology.

Keywords: Lattice-Based Cryptography · Side-Channel Attack · Higher-Order Masking · Fujisaki-Okamoto Transform

1 Introduction

The implementation-security aspect of candidates in the NIST Post-Quantum Standardization Process [NIS16] is becoming a focal point to enable widespread adoption of standardized algorithms. This is especially the case for the three finalist lattice-based Key Encapsulation Mechanisms (KEMs) Kyber [SAB+20], NTRU [CDH+20], and Saber [DKR+20], which, due to their good performance and acceptable memory footprint, are a fitting choice for use in embedded systems possibly exposed to side-channel attacks.

An interesting property of the aforementioned schemes is that they use a chosen-plaintext (CPA) secure public-key encryption scheme as a building block and then apply a CCA conversion to construct a chosen-ciphertext secure KEM. In particular, Kyber and Saber make use of variants of the Fujisaki-Okamoto (FO) transform [FO99]. In simplified

terms, the FO involves re-encrypting a decrypted message and then testing if the obtained ciphertext c' is equal to the input ciphertext c. Only in case of a match, the correct shared secret is released. If there is no match, the input ciphertext was malformed and a fixed non-sensitive value is returned.

The security of the FO transformation relies on the fact that no sensitive value is available to an adversary when a malformed ciphertext is inputted. While the FO protects against black-box attacks with this strategy, it does not protect against adversaries that use side-channels to learn sensitive intermediate values as shown in, e.g., [DTVV19, RRCB20, XPRO20, NDGJ21, UXT+21]. These works demonstrate that without appropriate protection there exist relatively simple (in terms of measurement requirements) yet still powerful attacks, and that virtually all operations in the ciphertext decapsulation need to be protected against such adversaries.

A particularly hard-to-protect operation in this regard is the test to verify if the recomputed ciphertext c' matches the input ciphertext c. While the final outcome of this comparison is not sensitive, a (side-channel) attacker must not learn *where* c and c' differ [BDH+21] in case the comparison fails. The first approach to mask the comparison of a CCA-secure scheme was presented in [OSPG18]. However, the idea is restricted to first-order masking. Subsequently, [BPO+20] presented a possibility to compare masked polynomials to public polynomials at arbitrary orders using a method we will refer to as the 'random sum' method. Lately, both techniques were broken as they make use of partial comparisons of the ciphertext [BDH+21]. These partial comparisons leak *where* c and c' differ, and this leakage can be used to mount key-recovery attacks on the schemes.

A straightforward fix to the first-order hash-based approach has already been introduced in a side-channel resistant implementation of Saber [VBDK+21] and has additionally been applied in a recent first-order masked Kyber implementation by Fritzmann et al. [FVBR+21]. The higher-order random sum method proves more difficult to fix. In [BDH+21], the authors analyze how the concepts of [BPO+20] can be salvaged to reduce the complexity of the comparison. However, this technique must still be integrated into a masked comparison method, and, importantly, it does not apply to schemes that compress the ciphertext, like Saber and Kyber.

At the moment, the only proper higher-order option (without any known security flaws) to perform a masked comparison of polynomials is the uncompressed range check by Bos et al. [BGR+21]. On the Cortex-M4, the method takes around 22 million cycles for second-order side-channel security (72 million for third-order), which is around 50% (63%) of the total execution time of the masked CCA-secure decapsulation. Therefore, the masked comparison can be considered a major bottleneck for higher-order secure implementations.

Contribution. In this work, we first show a higher-order side-channel attack that can easily break the first-order hash-based approach of [OSPG18] and its fixed variants [VBDK+21, BDH+21]. We recover the key using only 6000 measurements, which further emphasizes the need for new higher-order techniques for masked comparisons of (compressed) polynomials.

Subsequently, we present a novel technique to compare ciphertexts in higher-order masked schemes based on the (insecure) random sum method of [BPO+20]. In contrast to previous techniques we introduce a modulus switch to obtain a higher modulus and thus a lower collision probability, which results in only one equality check that needs to be performed. Furthermore, we show that it is possible to switch to a power-of-two modulus (in contrast with previous methods which only work for prime moduli). This power-of-two modulus integrates better with existing masked techniques such as B2A conversion.

Contrary to previous random sum methods, our method works for both schemes that have prime and power-of-two moduli coefficients and takes ciphertext compression into account. We show that it significantly outperforms the method of [BGR+21] for higher

orders and provide a security proof, correctness proof, as well as a masking security proof. Finally, we practically verify our results by providing the TVLA results of our implementation of the first- and second-order comparison using the state-of-the-art 'Fixed + Noise' versus Random (FNvR) framework from [BDH+21].

Outline. In Section 2, we recall Kyber and Saber, the Fujisaki-Okamoto transform, and previous proposals for masking the comparison operation. Then, in Section 3, we describe our attack on the first-order hash-based comparison method. We introduce our new masking approach and prove its security in Section 4. We then analyze its performance and verify the absence of leakage in Section 5. Finally, we conclude in Section 6.

2 Preliminaries

We now introduce some background on our target schemes Kyber and Saber, on the Fujisaki-Okamoto transform, which is the cause of requiring a comparison, and previous proposals aiming at securing said comparison.

2.1 Notation

We denote with $\lfloor x \rfloor$ flooring a number $x \in \mathbb{R}$ to the closest lower integer and with $\lfloor x \rceil$ rounding x to the nearest integer with ties rounded upwards. Furthermore, we write $y = \lfloor x \rceil_{q \to p}$ to denote $y = \lfloor (p/q) \cdot x \rceil$ for an input $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_p$. These operations are extended coefficient-wise for vectors and polynomials. For $x, q \in \mathbb{Z}$ we write $x \bmod q$ to denote the integer $\hat{x} \in (-q/2, q/2]$ so that $\hat{x} \equiv x \bmod q$. For a vector or polynomial x we denote with x_i taking the i^{th} coefficient of x, which is sometimes made more explicit as $x[i]$. We denote with $x \xleftarrow{\$} \chi$ sampling x randomly according to the distribution χ, and with $x \xleftarrow{r} \chi$ sampling pseudorandomly based on the seed r. Let $\mathcal{U}(I)$ denote the uniform distribution over a set I. A variable A that is masked into S shares is denoted in bold $\boldsymbol{A}$. When needed for clarity we distinguish between a Boolean masked variable $\boldsymbol{A}^B$ and arithmetic masked variable $\boldsymbol{A}^A$. We denote the j^{th} share with $\boldsymbol{A}^{(j)}$.

2.2 Saber and Kyber

The comparison technique presented in this paper is broadly applicable for masking implementations where two vectors need to be checked for equality. This can for example be necessary within the equality check of the FO transformation (see Section 2.3) of lattice-based Key Encapsulation Mechanisms (KEMs). In this paper, we will specifically target the use-case of masking implementations of IND-CCA secure KEMs Saber and Kyber.

Algorithms 1 to 3 give a generalized and simplified overview of the working of the IND-CPA secure encryption schemes Kyber and Saber. These can then be compiled to an IND-CCA secure KEM using the FO transformation as explained in Section 2.3. Both Kyber and Saber work with vectors of ring elements in R_q^k, where $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, with $n = 256$, and with k an integer between 2 and 4 depending on the security. The distributions $\chi(R_q^{k \times 1})$ generate vectors of polynomials with coefficients following a small binomial distribution. The modulus $q = q_2$ for Kyber is chosen as a prime, and the compression moduli p and T are powers of two. In Saber, all moduli q, q_2, p, T are powers of two, with $q > q_2 = p > T$. For a more in-depth discussion of Saber and Kyber, we refer to [DKRV18] and [BDK+18].

Algorithm 1: CPA.KeyGen.

$\text{sd}_A \xleftarrow{\$} \{0,1\}^{256}$;
$A \xleftarrow{\text{sd}_A} \mathcal{U}(R_q^{k\times k})$;
$(s,e) \xleftarrow{\$} \chi(R_q^{k\times 1}) \times \chi(R_q^{k\times 1})$;
$t \leftarrow \lfloor A \cdot s + e \rceil_{q\to q_2}$;
return $pk := (sd_A, t), sk := s$;

Algorithm 2: CPA.Enc.

Input: $pk = (\text{sd}_A, t)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
$A \xleftarrow{\text{sd}_A} \mathcal{U}(R_q^{k\times k})$;
$(r, e_1, e_2) \xleftarrow{r} \chi(R_q^{k\times 1}) \times \chi(R_q^{k\times 1}) \times \chi(R_q^{1\times 1})$;
$u \leftarrow A \cdot r + e_1$;
$v \leftarrow (\frac{q}{q_2} \cdot t) \cdot r + e_2 + \lceil \frac{q}{2} \rfloor \cdot m$;
$c_1 \leftarrow \lfloor u \rceil_{q\to p}$;
$c_2 \leftarrow \lfloor v \rceil_{q\to T}$;
return $c := (c_1, c_2)$;

Algorithm 3: CPA.Dec.

Input: $sk = s$
Input: $c = (c_1, c_2)$
$u \leftarrow \lfloor c_1 \rceil_{p\to q}$;
$v \leftarrow \lfloor c_2 \rceil_{T\to q}$;
$m \leftarrow \lfloor v - s^T \cdot u \rceil_{q\to 2}$;
return m ;

Algorithm 4: CCAKEM.KeyGen.

$z \xleftarrow{\$} \{0,1\}^{256}$;
$(pk, sk') = \text{CPA.KeyGen}()$;
$sk = (sk'||pk||\text{H}(pk)||z)$;
return pk, sk ;

Algorithm 5: CCAKEM.Encaps.

Input: Public key of CCAKEM pk
$m \xleftarrow{\$} \{0,1\}^{256}$;
$m \leftarrow \text{H}(m)$;
$(\bar{K}, r) = \text{G}(m||\text{H}(pk))$;
$c = \text{CPA.Enc}(pk, m, r)$;
$K = \text{KDF}(\bar{K}||\text{H}(c))$;
return c, K ;

Algorithm 6: CCAKEM.Decaps.

Input: Ciphertext of CCAKEM c
Input: Secret key of CCAKEM sk
Extract $(sk'||pk||\text{H}(pk)||z)$ from sk ;
$m' = \text{CPA.Dec}(sk', c)$;
$(\bar{K}', r') = \text{G}(m'||\text{H}(pk))$;
$c' = \text{CPA.Enc}(pk, m', r')$;
if $c = c'$ **then**
  $K = \text{KDF}(\bar{K}'||\text{H}(c))$;
else
  $K = \text{KDF}(z||\text{H}(c))$;
end
return K ;

2.3 FO-transformation

The FO transformation [FO99, HHK17] is a generic method to convert an IND-CPA secure encryption scheme into an IND-CCA secure KEM. In the FO-transformation, the encapsulation is a deterministic version of the encryption, where all randomness is pseudorandomly based on the message (which is itself chosen at random). This means that one can exactly recompute the ciphertext using the message. The high-level idea is that during decapsulation, the message is decrypted and then used to check if the inputted ciphertext is well-formed. This check is performed by re-encrypting the message and checking if the input ciphertext is equal to the re-encrypted message. An overview of the FO transformation is given in Algorithms 4 to 6, where G and H represent hash functions and where KDF is a key derivation function.

When the input ciphertext is invalid, the decapsulation will return unusable randomness so that an adversary does not gain information (except for the fact that the ciphertext was rejected). The security of the FO transformation relies on the fact that in this case, an attacker does not learn anything about the intermediate values of the computation. Protection of these routines against side-channel attacks is, therefore, of utmost importance.

2.4 Masked Comparison Algorithms

2.4.1 Hash-Based Method [OSPG18]

In [OSPG18], an efficient method for constructing a first-order masked ciphertext comparison is presented. While the original method has a subtle flaw [BDH+21], there exists an easy fix, and the general method was used in several secured implementations of e.g., Saber and Kyber [FVBR+21, VBDK+21].

Assume that the re-encryption outputs its result c' in two Boolean shares $(\mathbf{c}'^{(0)}, \mathbf{c}'^{(1)})$, i.e., $c' = \mathbf{c}'^{(0)} \oplus \mathbf{c}'^{(1)}$. To test if c is equal to c', [OSPG18] propose to compute

$$\mathrm{H}(c \oplus \mathbf{c}'^{(0)}) \stackrel{?}{=} \mathrm{H}(\mathbf{c}'^{(1)}). \tag{1}$$

Only if $c = c'$, the two hash function calls receive the same input and the hashes match. Due to the random sharing of c', the actual inputs are randomized for each decapsulation. A drawback of this method is that it only works for first-order maskings.

2.4.2 Random Sum Method [BPO+20] [BDH+21]

Bache et al. [BPO+20] introduced a new method to allow higher-order masking of the comparison operation. Bhasin at al. [BDH+21] later showed some vulnerabilities in this method and provided a rough idea of how to solve these vulnerabilities. Both only describe the case where the modulus is prime and where there is no ciphertext compression. In this paper, we will generalize to both, prime and power-of-two moduli, and to schemes that undergo ciphertext compression.

The core idea for of the random sum method is to test if a list of input coefficients are all zero: given n masked coefficients in S shares $\boldsymbol{D}_0, \cdots, \boldsymbol{D}_{n-1} \in \mathbb{Z}_q^S$ we want to calculate for every i, if the sensitive unmasked sum $\sum_j \boldsymbol{D}_i^{(j)}$ is zero.

Instead of performing the zero check for all coefficients individually, we compress them in one term. However, to avoid giving away sensitive information we first want to do this separately for each share, by calculating $\boldsymbol{E}^{(j)} = \sum_i R_i \boldsymbol{D}_i^{(j)}$ for each share[1]. In a second phase one then checks if $\sum_j \boldsymbol{E}^{(j)} = 0$. On one hand, if every unmasked coefficient $\sum_j \boldsymbol{D}_i^{(j)} = 0$ then clearly $\sum_j \boldsymbol{E}^{(j)} = 0$. On the other hand if at least one of the coefficients is not zero, the term $\sum_j \boldsymbol{E}^{(j)}$ will only be zero with limited probability (depending on the random R_i's).

For a prime q, one can show that the probability of such a false positive response is equal to the probability that a random element of $\mathbb{Z}_q$ is zero, which is $1/q$. For a power-of-two q, an adversary can choose his input coefficients as $q/2$ or 0 to increase this false positive probability to $1/2$. As in both cases, the straightforward scenario does not give enough certainty to obtain security in typical cryptographic applications, it is proposed in [BPO+20] to replicate the check L times, to limit the false positive probability to $1/q^L$ (or $1/2^L$ times for power-of-two q). In [BPO+20] this is done by dividing the coefficients into k sets and performing the check on each subset individually.

In [BDH+21] two vulnerabilities in the above repetition method were shown: Firstly, leakage of intermediate results of the check results leads to a first-order side-channel attack and must therefore be avoided. Secondly, performing the intermediate checks on only a subset of the coefficients leads to a chosen-ciphertext attack where the adversary inputs a slightly adapted ciphertext that only fails in one of the L intermediate checks, increasing the false positive probability to $1/q$ (or $1/2$ for power-of-two q). The authors concluded

[1] The original method [BPO+20] calculated $\boldsymbol{E}^{(j)} = \sum_i R_{0i}(\boldsymbol{D}_i^{(j)} + R_{1i})$. Bhasin at al. [BDH+21] showed that one does not need the second randomness R_{1i}. In this paper we will for sake of clarity discuss their simplified method.

that a random sum algorithm must not leak any results of the intermediate checks and that it must always be calculated over all coefficients.

In this paper, we construct a random sum-based technique that deals with the problems in the original technique by moving to a large q. This reduces the false-negative probability, which allows relying only on one check ($L = 1$). More information about our technique will be given in Section 4.

3 Side-Channel Collision Attack on a 1st-Order Impl.

In this section, we present a side-channel attack that can easily break the first-order hash-based comparison method of [OSPG18]. Our attack belongs to the category of horizontal collision attacks [MME10], i.e., side-channel leakage is used to test if the same data is processed at two selected points during computing $\mathrm{H}(c \oplus \mathbf{c'}^{(0)}) \stackrel{?}{=} \mathrm{H}(\mathbf{c'}^{(1)})$. As this can be considered a second-order attack, no security claims of the masking approach are broken. Still, the attack is unprofiled, requires only minimal knowledge of the implementation, is likely robust in terms of noise, and for these reasons easy to perform. Hence, it shows that the hash-based first-order approach and likely also other first-order approaches using Boolean masking do not offer sufficient protection.

3.1 Attack Description

We use Kyber for all our explanations and experiments, but we note that the hash-based comparison approach, and hence our attack, is also applicable to Saber. Our attack follows along the lines of the generic side-channel attack using a decryption failure oracle described in [BDH+21]. That is, we honestly generate a ciphertext and then manipulate a single coefficient of the second ciphertext component c_2 (corresponding to v) while keeping the first component c_1 (corresponding to u) untouched. Depending on the concrete value of the secret key, this manipulation can lead to decryption failure, i.e., the recovered message m' can differ from the m used during encapsulation in a single bit.

Since the random coins r used for re-encryption are derived by hashing m', this single-bit error leads to entirely different values being used during re-encryption and thus a ciphertext c' having no resemblance of c. If, however, no decryption failure occurs, then c and c' differ only in a couple of bits (one coefficient of c_2). While these two scenarios are not discernable in a black-box setting (both lead to the ciphertext being rejected), they can be distinguished using side-channel measurements, which in turn gives information on the secret key.

Concretely, we use that when back-substituting decryption, we get

$$v - s^T u = \left\lceil \frac{q}{2} \right\rfloor \cdot m + e^T r - s^T(e_1 + \Delta u) + e_2 + \Delta v, \tag{2}$$

with Δu and Δv denoting the error introduced by compression of the two ciphertext components. We call

$$d = e^T r - s^T(e_1 + \Delta u) + e_2 + \Delta v \tag{3}$$

the decryption noise. The parameters of Kyber are chosen such that $||d||_\infty < q/4$ with very high probability to ensure that no decryption errors occur. Since all elements of d are in some form small and the range is limited, d can be lifted to $\mathbb{Z}$.

For the attack, we honestly generate a ciphertext and then add $q/4$ to one selected coefficient of v.[2] The corresponding message bit at index i will still be correctly decoded

[2] Due to ciphertext compression, the actual value might differ slightly from $q/4$.

only if $d_i + q/4 < q/4$, i.e., if $d_i < 0$. If, however, $d_i \geq 0$, then a decryption error will occur.[3]

Detecting decryption errors. We use the following method to detect decryption errors in the first-order hash-based masked comparison. As explained in Section 2.4.1 during the comparison the two masked ciphertext components are hashed and their equality is checked as $\mathrm{H}(c \oplus \mathbf{c'}^{(0)}) \stackrel{?}{=} \mathrm{H}(\mathbf{c'}^{(1)})$.

First, we note that for Kyber, c is at least 768 bytes large, which is larger than the block size of the hash function. For example, when initiating H with SHAKE128, which has a block size 1344 bits, c is split into at least 5 blocks. Moreover, the ciphertext consists of two parts $c = (c_1||c_2)$. Since c_1 is typically larger than c_2, we have that c_2 is only part of the last input blocks (in our case 2), while all previous blocks only contain c_1.

We can now differentiate based on whether a decryption error occurs or not. Remember that in case of a decryption error, c' is essentially independent of c as it is generated using entirely different coins r'. In this case, the inputs to $\mathrm{H}(c \oplus \mathbf{c'}^{(0)})$ and $\mathrm{H}(\mathbf{c'}^{(1)})$ differ already in the first block. If, however, the correct message is still recovered, then c and c' differ only in a few bits of c_2 in the last hash blocks. Followingly, the first couple of blocks of $\mathrm{H}(c \oplus \mathbf{c'}^{(0)})$ and $\mathrm{H}(\mathbf{c'}^{(1)})$ use identical input.

Thus, by using a side-channel collision attack on the first blocks of $\mathrm{H}(c \oplus \mathbf{c'}^{(0)})$ and $\mathrm{H}(\mathbf{c'}^{(1)})$, e.g., comparing the power consumption and determine if they are similar, one can determine if a decryption failure occurred, and followingly, the sign of d_i. This collision attack can use a large portion of the trace (at least 3 full Keccak-f permutations), which is why a single trace is usually sufficient and the noise robustness is high.

Solving for the key. After gathering many traces and extracting the sign of the respective d_i, one needs to extract the key from this information. In [BDH+21], the relation $(v - us)_i \approx m_i \cdot \lceil \frac{q}{2} \rfloor + d_i$[4] is fed to the *Leaky-LWE* framework of Dachman-Soled et al. [DDGR20]. In this framework, the lattice described by the public-key equation $t = As + e$ is transformed using hints gathered via side channels. We found that this approach is not ideal for the problem at hand: the runtime needed for including the hints in the lattice is quite high and the security level decreases very slowly with the number of traces. For instance, after gathering 2^{17} approximate equations, each requiring multiple measurements, [BDH+21] still report a key-recovery complexity of roughly 2^{64} operations for Kyber512.

Recently, [PP21] and [HPP21] presented fault attacks on several CCA-secure lattice-based KEMs, including Kyber. Their attacks can be classified as safe-error attacks [YJ00] in that they inject a specific fault and then observe if decapsulation still returns the correct result. As it turns out, their key-recovery problem is identical to ours. They propose a different solving approach, which we now briefly describe.

Their approach exploits the composition of d, i.e., the structure of the right-hand side of eq. (3). Note that if the attacker honestly generated the ciphertext by running encapsulation using the public key, then only the secret key (e, s) is unknown; all other values in d are also generated or can be computed during encapsulation. Since d_i is small, we thus have that d_i is linear (in $\mathbb{Z}$) in the key coefficients. Since the side-channel attack only extracts the sign of d_i, the equalities turn into *inequalities* of form

$$e^T r - s^T(e_1 + \Delta u) + e_2 + \Delta v \lessgtr 0. \tag{4}$$

[3] In [BDH+21], the range of d_i was further narrowed down using a binary search over all $2^T = 16$ possible values of c_2, performing a measurement in each step. However, since $|d| < q/16$ (which is the smallest possible increment of v due to compression to c_2) in the vast majority of cases, using a new ciphertext for each measurement appears to leak more information on the key compared to using multiple measurements on a single base ciphertext.

[4] Approximation since d_i is only known up to an interval of size $q/16$.

After gathering enough inequalities, they solve for the key using an approach akin to linear decoding. We employ the improved method of [HPP21], which reaches a success rate close to 1 when using roughly 5750, 6750, and 8500 inequalities (and thus measurements) for Kyber512, Kyber768, and Kyber1024, respectively.

3.2 Attack Setup and Measurement

We verified the correctness of our attack by attacking a microcontroller running Kyber. Our target is an STM32F405 (ARM Cortex-M4) mounted on a ChipWhisperer side-channel evaluation board [Newb]. The microcontroller was clocked at 24 MHz to match the frequency used in the popular PQM4 PQC benchmarking framework [KRSS]. We measured the voltage drop over the on-board shunt resistor using a LeCroy AP034 differential probe. Due to the length of the measurements (multiple Keccak-f permutations per trace) and the limited sample memory of the ChipWhisperer Lite platform [Newa], we performed measurements using an oscilloscope sampling at 100 MS/s. To reduce noise, we used the scopes in-built 20 MHz analog filter.

The microcontroller runs the most recent ASM-optimized Kyber512 implementation included in PQM4 [KRSS]. This implementation is unprotected; we added the first-order masked comparison as described above, where we used the included ASM-optimized SHAKE128 for H. A trigger signal is set to mark the start of the masked comparison, i.e., the computation of the hash functions. We generated the manipulated ciphertexts on a PC (using the known public key) and then send them to the device for decapsulation.

Trace processing. Each trace contains two invocations of H, namely $\mathrm{H}(c \oplus \mathbf{c}'^{(0)})$ and $\mathrm{H}(\mathbf{c}'^{(1)})$. To localize these calls and align them for the horizontal side-channel attack, we compute an autocorrelation over the trace and then select its peak index as the beginning of the second subtrace. This method can also be used to find the hashes if no dedicated trigger signal is available.

We then perform a pointwise subtraction of the two trace segments and compute the mean of the squared difference. If this quantity is above a certain threshold, we conclude that the two hash calls processed vastly different inputs, i.e., a decryption error occurred. If the score is below the threshold, then the message m' was correctly recovered and the first input blocks of the two hashes are identical. This principle is illustrated in Figure 1, which shows collision scores obtained by adding all $2^T - 1 = 15$ possible offsets to one coefficient c_2 (compressed v).

We dynamically determine the threshold by sending two modified ciphertexts and taking the midway point of the mean-squared difference as threshold. The first ciphertext is random, hence a decryption error will occur and the hashes will have different inputs. The second ciphertext is honestly generated, but we add 1 to one coefficient of c_2. It is highly unlikely that this change leads to a decryption error, the hash inputs thus differ only in 1 bit in one of the later blocks.

3.3 Results

We performed a total of 10 experiments, each using a different key. We collected 6000 measurements per experiment, which, according to [HPP21], should suffice for key recovery. All 10 experiments were successful; we only observed 2 misclassified d_i throughout all 60 000 measurements.

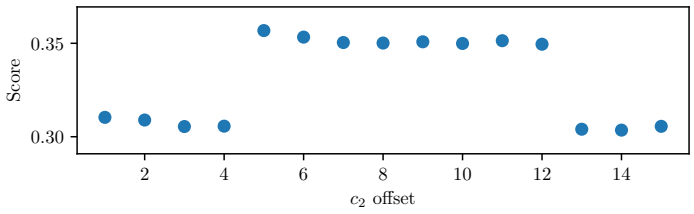


Figure 1: Values of the "collision score" obtained by adding "c_2 offset" mod 2^T to one coefficient of c_2 (keeping the ciphertext the same on all other positions). Adding values between 5 and 12 lead to a decryption error, all other offsets do not. The two cases can be easily distinguished.

4 Higher-Order Masked Comparison

In this section, we describe our new higher-order masked comparison. We start with explaining our method and continue with three security proofs, showing correctness, security, and masking security. Contrary to previous random sum methods, our method works for both schemes that use a power-of-two and prime modulus and so accommodates both Saber and Kyber.

As explained in Section 2.4.2, the random sum method has a false-positive probability in which the method wrongly thinks all input coefficients equal zero. In both [BPO+20] and [BDH+21], false positives happen with a non-negligible probability, and these works proposed to perform multiple checks to further reduce this probability.

In this work, we want to perform only one check. We achieve this by enlarging the masking modulus q up to a point where the false positive probability is sufficiently small. While previous works only considered a prime modulus as it avoids zero divisors that lead to a higher false positive probability, we prove that in our design it is also possible to use a power-of-two modulus. We specifically choose a large power-of-two modulus as it interplays well with other masking techniques such as Arithmetic to Boolean (A2B) or Boolean to Arithmetic (B2A) mask conversion. However, it would also be possible to choose a large prime as new modulus. We specifically focus our method on lattice-based encryption schemes with compression (e.g., Saber [DKRV18] or Kyber [BDK+18]) which require additional preprocessing of the inputs.

Algorithm 7 gives an overview of our technique. It relies on three subfunctions: `A2B`, which transforms an arithmetic sharing into a Boolean sharing, $\texttt{B2A}_{p\to q}$, which transforms a p bit Boolean masking to a q bit arithmetic masking, and `BooleanEqualityTest`, which tests if a sharing of a single coefficient $\boldsymbol{E}$ represents zero (i.e., $\sum_S \boldsymbol{E}^{(j)} = 0$).

The algorithm consists of five steps. Step 1 and 2 convert the input from a small modulus to a larger modulus $p \cdot 2^{s-1}$ where s is a security parameter that will determine the false positive probability (which will equal 2^{-s}). In step 3 we perform the compression as proposed in [BPO+20] and improved in [BDH+21] and step 4 performs a check on only one (large) coefficient.

Step 0 does an application-specific preprocessing. For our case this consists of three parts: First, in the case of lattice-based encryption, we actually need to compare two coefficients (a masked one with a public one) instead of performing a zero check. The addition of the constant $2^{f_{bits,B}}/2$ is used to mimic a rounding operation when shifting right in step 1. By subtracting the public coefficients from the first share of each corresponding masked coefficient (lines 5 and 11) we convert the comparison to a zero check.

Secondly, for non power-of-two moduli q we scale to a power-of-two modulus to make the conversion to the larger modulus $p \cdot 2^{s-1}$ easier (line 3 and line 9). To avoid introducing errors during rounding we take into account a sufficient number of fractional bits. In

Algorithm 7: Masked Comparison

Input : masked vector of elements $\boldsymbol{B}^* \in \mathbb{Z}_q^{kn,(S)}$, vector of elements $B \in \mathbb{Z}_p^{kn}$;
masked vector of elements $\boldsymbol{C}^* \in \mathbb{Z}_q^{n,(S)}$, vector of elements $C \in \mathbb{Z}_T^n$;
with p, T powers of two and $p \geq T$

Output : $\begin{cases} 1 & \text{if: } \left(\forall_i \lfloor \frac{p}{q} \sum_j \boldsymbol{B}_i^{*,(j)} \rceil = B_i \bmod p\right) \text{and } \left(\forall_i \lfloor \frac{p}{q} \sum_j \boldsymbol{C}_i^{*,(j)} \rceil = C_i \bmod T\right) \\ 0 & \text{else} \end{cases}$

// step 0: preprocessing
for $i = 0$ **to** $kn - 1$ **do**
  for $j = 0$ **to** $S - 1$ **do**
    $\boldsymbol{B}_i^{A,(j)} = \lfloor \frac{p \cdot 2^{f_{bits,B}}}{q} \boldsymbol{B}_i^{*,(j)} \rfloor$
  end
  $\boldsymbol{B}_i^{A,(0)} = \left(\boldsymbol{B}_i^{A,(0)} - 2^{f_{bits,B}} \cdot B_i + 2^{f_{bits,B}}/2\right) \bmod p \cdot 2^{f_{bits,B}}$
end
for $i = 0$ **to** $n - 1$ **do**
  for $j = 0$ **to** $S - 1$ **do**
    $\boldsymbol{C}_i^{A,(j)} = \lfloor \frac{T \cdot 2^{f_{bits,C}}}{p} \boldsymbol{C}_i^{*,(j)} \rfloor$
  end
  $\boldsymbol{C}_i^{A,(0)} = \left(\boldsymbol{C}_i^{A,(0)} - 2^{f_{bits,C}} \cdot C_i + 2^{f_{bits,C}}/2\right) \bmod T \cdot 2^{f_{bits,C}}$
end

// step 1: convert to Boolean masking
for $i = 0$ **to** $kn - 1$ **do**
  $\boldsymbol{B}_i^B = \texttt{A2B}(\boldsymbol{B}_i^A) \gg f_{bits,B}$
end
for $i = 0$ **to** $n - 1$ **do**
  $\boldsymbol{C}_i^B = \texttt{A2B}(\boldsymbol{C}_i^A) \gg f_{bits,C}$
end

// step 2: convert to arithmetic masking modulo $p \cdot 2^{s-1}$
for $i = 0$ **to** $kn - 1$ **do**
  $\boldsymbol{D}_i = \texttt{B2A}_{p \to p \cdot 2^{s-1}}(\boldsymbol{B}_i^B)$
end
for $i = 0$ **to** $n - 1$ **do**
  $\boldsymbol{D}_{i+kn} = \texttt{B2A}_{T \to p \cdot 2^{s-1}}(\boldsymbol{C}_i^B)$
end

// step 3: combine all checks in one term
$\boldsymbol{E} = 0$
for $i = 1$ **to** $(k+1)n$ **do**
  $R_i \xleftarrow{\$} \mathcal{U}(\,[0, 2^s)\,)$
  for $j = 0$ **to** $S - 1$ **do**
    $\boldsymbol{E}^{(j)} = \boldsymbol{E}^{(j)} + R_i \cdot \boldsymbol{D}_i^{(j)} \bmod p \cdot 2^{s-1}$
  end
end

// step 4: Test if $\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1}$
return BooleanEqualityTest($\boldsymbol{E}$)

practice we have to choose:

$$f_{bits,B} = f_{bits,C} > \log_2(S) - \log_2\left(\frac{\lceil q/2 \rceil}{q} - \frac{1}{2}\right).$$

For example, $f_{bits,B} = f_{bits,C} = 13$ for masked Kyber with two shares as discussed in [FVBR+21]). This step is ignored for power-of-two schemes, where $2^{f_{bits,B}} = q/p$ and $2^{f_{bits,C}} = T/p$.

Finally, we need to perform compression from modulus q to a modulus p or T. This is performed in line 14 and line 17 as this operation is trivial to do in the Boolean domain. Note that previous random sum comparison methods [BPO+20, BDH+21] were not capable of handling the compression of the ciphertext.

As with previous random sum methods, our method is prone to false positives (also named collisions in [BDH+21]) with a small probability of 2^{-s}. When such a collision occurs, a non-valid ciphertext is accepted and as shown in [BDH+21] an adversary can use these collisions to reduce the security of the attacked scheme.

An adversary needs to submit on average 2^s invalid ciphertexts to obtain one collision. In our security proof we will show that this collision probability is independent of the input of an adversary and as such an adversary can not increase the probability of triggering a collision. Therefore techniques like failure boosting [DGJ+19] to reduce the failure probability are not applicable in this context. Moreover, multiple collisions would be required to significantly reduce the security of the scheme.

The security parameter s can be chosen at any arbitrary value depending on the application scenario (at the cost of additional operations and randomness). In our implementation, we choose $s = 54$ so that the maximal bitwidth of the variables is 64 bits. This corresponds to a collision probability of 2^{-54} or an expected 2^{54} queries that an adversary needs to do to obtain one collision. We provide additional implementations with $s = 118$ (maximal bitwidth 128 bits) and $s = 128$ (maximal bitwidth 138 bits) in Appendix A.

4.1 Security Proof

Theorem 1 (Correctness and Security of Algorithm 7)**.** *Let p, T be powers of two, let q be a power of two or a prime, and let k, n, s be integers. If q is prime then we require $f_{bits,B}$ and $f_{bits,C}$ to be larger than $\log_2(S) - \log_2\left(\frac{\lceil q/2 \rceil}{q} - 0.5\right)$. If q is a power of two then $2^{f_{bits,B}} = q/p$ and $2^{f_{bits,C}} = T/p$. Then upon input $\boldsymbol{B}^* \in \mathbb{Z}_q^{kn,(S)}$, $B \in \mathbb{Z}_p^{kn}$, $\boldsymbol{C}^* \in \mathbb{Z}_q^{n,(S)}$, $C \in \mathbb{Z}_T^n$, Algorithm 7 returns:*

- *1 if: $\left(\forall_{i=0}^{kn-1} \lfloor \frac{p}{q} \sum_{j=0}^{S-1} \boldsymbol{B}_i^{*,(j)} \rceil = B_i \bmod p\right)$ and $\left(\forall_{i=0}^{n-1} \lfloor \frac{T}{q} \sum_{j=0}^{S-1} \boldsymbol{C}_i^{*,(j)} \rceil = C_i \bmod T\right)$*
- *0 with probability at least $1 - 2^{-s}$ if the above condition is not fulfilled*

Proof. We will start with proving correctness, i.e. showing that the output will be 1 if the conditions are met. Our proof follows the strategy of [BDH+21] using the power-of-two technique of [CDE+18].

By simple substitution we have:

$$\sum_j \boldsymbol{E}^{(j)} \bmod p \cdot 2^{s-1} = \sum_j \sum_i R_i \boldsymbol{D}_i^{(j)} \bmod p \cdot 2^{s-1} \tag{5}$$

$$= \sum_i R_i \cdot \sum_j (\boldsymbol{D}_i^{(j)} \bmod p \cdot 2^{s-1}) \bmod p \cdot 2^{s-1} \tag{6}$$

$$= \begin{pmatrix} \sum_{i=0}^{kn-1} R_i \cdot (\bigoplus_j \boldsymbol{B}_i^{B,(j)}) \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot (\bigoplus_j \boldsymbol{C}_i^{B,(j)}) \end{pmatrix} \bmod p \cdot 2^{s-1} \tag{7}$$

We continue by focussing on the term $\bigoplus_j \boldsymbol{B}_i^{B,(j)}$, which can be further expanded to:

$$\bigoplus_j \boldsymbol{B}_i^{B,(j)} = \left\lfloor \sum_j \boldsymbol{B}_i^{A,(j)} / 2^{f_{bits,B}} \right\rceil \bmod p \tag{8}$$

$$= \left\lfloor \sum_j \lfloor \frac{p \cdot 2^{f_{bits,B}}}{q} \boldsymbol{B}_i^{*,(j)} \rfloor / 2^{f_{bits,B}} + \frac{1}{2} \right\rfloor - B_i \bmod p \tag{9}$$

For power-of-two moduli where $2^{f_{bits,B}} = q/p$ the inner flooring operates on integers and can therefore be ignored. For non power-of-two moduli q we used the trick as described in [FVBR+21]. In this case we can rewrite the equation as:

$$= \left\lfloor \sum_j \left(\frac{p \cdot 2^{f_{bits,B}}}{q} \boldsymbol{B}_i^{*,(j)} - e^{(j)} \right) / 2^{f_{bits,B}} + \frac{1}{2} \right\rfloor - B_i \bmod p \tag{10}$$

$$= \left\lfloor \sum_j \frac{p}{q} \boldsymbol{B}_i^{*,(j)} - e^{(j)} / 2^{f_{bits,B}} + \frac{1}{2} \right\rfloor - B_i \bmod p, \tag{11}$$

where $e^{(j)}$ is the flooring error that can be bounded by $0 \leq e^{(j)} < 1$.

Note that we can drop the error term as long as this does not produce an overflow, that is $\lfloor y - e + \frac{1}{2} \rfloor = \lfloor y + \frac{1}{2} \rfloor$ as long as $e < (y + \frac{1}{2} \mod 1)$. On one hand we can see that $y + \frac{1}{2} \mod 1$ has only a limited number of possible values, which can be decribed as the set $\{i/q + \frac{1}{2} \mod 1 \mid i \in [0, q)]\}$ (as described in [FVBR+21]). The worst case scenario is with $i = \lceil q/2 \rceil$, in which case we have that $y + \frac{1}{2} \mod 1 = \frac{\lceil q/2 \rceil}{q} - \frac{1}{2}$. On the other hand, we have a worst case error value $e^{(j)} = 1$, so that $e < S/2^{f_{bits,B}}$. Thus, to remove the error term we need to have that:

$$e < (y + \frac{1}{2} \mod 1) \quad \text{or:} \tag{12}$$

$$S/2^{f_{bits,B}} < \frac{\lceil q/2 \rceil}{q} - \frac{1}{2} \quad \text{or:} \tag{13}$$

$$f_{bits,B} > log_2(S) - \log_2(\frac{\lceil q/2 \rceil}{q} - \frac{1}{2}). \tag{14}$$

Since we chose $f_{bits,B}$ to fulfill this condition, we can remove the error and write:

$$\bigoplus_j \boldsymbol{B}_i^{B,(j)} = \left\lfloor \sum_j \frac{p}{q} \boldsymbol{B}_i^{*,(j)} \right\rceil - B_i \bmod p \tag{15}$$

$$\tag{16}$$

An analogous derivation is also possible for the $\boldsymbol{C}$ terms. For sake of convenience we will denote $\beta_i = \left\lfloor \sum_j \frac{p}{q} \boldsymbol{B}_i^{*,(j)} \right\rceil - B_i \bmod p$ and similarly $\gamma_i = \left\lfloor \sum_j \frac{T}{q} \boldsymbol{C}_i^{*,(j)} \right\rceil - C_i \bmod T$. This results in the following equality:

$$\sum_j \boldsymbol{E}^{(j)} \bmod p \cdot 2^{s-1} = \begin{pmatrix} \sum_{i=0}^{kn-1} R_i \cdot \beta_i \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot \gamma_i \end{pmatrix} \bmod p \cdot 2^{s-1} \tag{17}$$

Correctness When the conditions are met, i.e. $\left(\forall_i \lfloor \frac{p}{q} \sum_j \boldsymbol{B}_i^{*,(j)} \rceil = B_i \bmod p\right)$ and $\left(\forall_i \lfloor \frac{T}{q} \sum_j \boldsymbol{C}_i^{*,(j)} \rceil = C_i \bmod T\right)$, all terms β_i and γ_i equal zero modulo p and T respectively. Furthermore, they are in $[-p/2, p/2)$ and $[-T/2, T/2)$ respectively and thus:

$$\sum_j \boldsymbol{E}^{(j)} \bmod p \cdot 2^{s-1} = 0$$

Security When the conditions are not met, at least one of the coefficients of B or C does not equal the corresponding masked value of $\boldsymbol{B}^*$ or $\boldsymbol{C}^*$. We will first look at the case where at least one of the coefficients of B does not match its corresponding value, and treat the case where only coefficient(s) of C differ later.

Without loss of generality, we can assume that a non-corresponding coefficient is located at the zeroth coefficient ($i = 0$). The output of Algorithm 7 is binary and the (incorrect) return value 1 is given when $\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1}$. We will first derive an equivalent condition in terms of the inputs and then show that due to the randomness of R_0, it is hard to guess an input that gives returns 1 when the condition is not fulfilled.

Rewriting $\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1}$ using Equation 17 we get:

$$\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1} \tag{18}$$

$$\iff \begin{pmatrix} \sum_{i=0}^{kn-1} R_i \cdot \beta_i \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot \gamma_i \end{pmatrix} = 0 \bmod p \cdot 2^{s-1} \tag{19}$$

$$\iff \begin{pmatrix} R_0 \cdot \beta_0 \\ + \sum_{i=1}^{kn-1} R_i \cdot \beta_i \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot \gamma_i \end{pmatrix} = 0 \bmod p \cdot 2^{s-1} \tag{20}$$

Now we choose z so that $2^z = \texttt{gcd}(p \cdot 2^{s-1}, \beta_0)$. Notice that $2^z \leq p/2$ as β_0 is strictly smaller than p. Moreover, when an incorrect response 1 is given, Equation 20 is satisfied and thus 2^z divides

$$\begin{matrix} \sum_{i=1}^{kn-1} R_i \cdot \beta_i \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot \gamma_i \end{matrix} .$$

We can then define the variables X and Y as follows:

$$X \leftarrow \beta_0 / 2^z \text{ and } Y \leftarrow \begin{pmatrix} \sum_{i=1}^{kn-1} R_i \cdot \beta_i \\ + \sum_{i=0}^{n-1} R_{kn+i} \cdot \gamma_i \end{pmatrix} / 2^z \tag{21}$$

To rewrite the condition (Equation 20) further as:

$$\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1} \tag{22}$$

$$\iff R_0 X 2^z + Y 2^z = 0 \bmod p \cdot 2^{s-1} \tag{23}$$

$$\iff R_0 X + Y = 0 \bmod p \cdot 2^{s-1-z} \tag{24}$$

$$\iff R_0 = -Y \cdot X^{-1} \bmod p \cdot 2^{s-1-z} \tag{25}$$

Where in the latter step we use the fact that $1 = \texttt{gcd}(X, p \cdot 2^{s-1-z})$ to conclude that X has an inverse. Notice that the modulus of this expression is $p \cdot 2^{s-1-z}$ and we know that $2^z \leq p/2$, from which we can conclude that the modulus $p \cdot 2^{s-1-z} \geq 2^s$ and thus R_0 retains its entropy in this expression.

As R_0 is independent of the terms X and Y and is unknown to any adversary, the probability of obtaining $\sum_j \boldsymbol{E}^{(j)} = 0 \bmod p \cdot 2^{s-1}$ when the condition is not met is upper-bounded by the guessing probability of R_0 which is 2^{-s}.

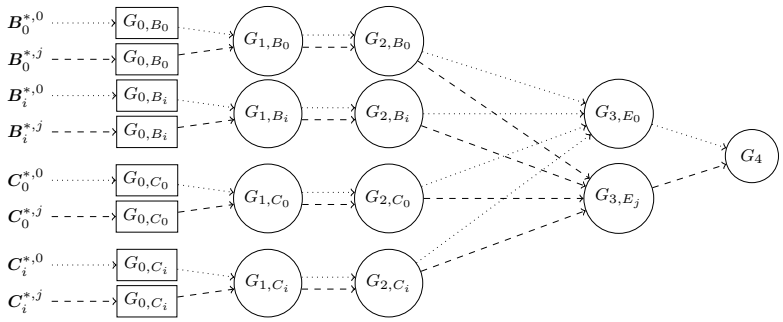


Figure 2: Overview of Gadgets in Algorithm 7. Different line types correspond to different shares.

A similar reasoning can be constructed if only coefficients of C are incorrect. The only difference is that 2^z is upper-bounded as $2^z \leq T/2$ due to the different modulus. However, as $T \leq p$ we can state $2^z \leq p/2$. The rest of the proof is analogous to the one outlined above. □

Theorem 2 (t-(S)NI of Algorithm 7)**.** *Let* `BooleanEqualityTest` *be a t-NI gadget and let* `A2B` *and* `B2A` *be t-SNI gadgets. Let* $\boldsymbol{B}^*$ *and* $\boldsymbol{C}^*$ *be the masked inputs of Algorithm 7. For any set of* $t_c \leq t$ *intermediate variables there exists subsets* I_B *and* I_C *of input indices with* $|I_B| + |I_C| \leq t_c$ *such that the* t_c *intermediate values can be perfectly simulated from the input variables* $\boldsymbol{B}^{*,(I_B)}$ *and* $\boldsymbol{C}^{*,(I_C)}$*. This t-NI security also implies t-SNI security as there is no sensitive output value.*

Proof. We divide Algorithm 7 into five types of gadgets representing the five steps in the algorithm. The first three types of gadgets G_0 to G_2 work on the coefficients individually, outputting variables with S shares. Gadget G_3 combines all coefficients, but works on each share individually (different shares are indicated with different arrow types in Figure 2).

Each gadget is subdivided into separate subgadgets that perform the same operation on different input data. An overview is given in Figure 2. The exact definition of the gadgets is:

Table 1: Overview of gadgets and their relation to the lines in Algorithm 7.

G_0	Step 0	$G_{1,B_i^{(j)}}$	line 3 (and line 5 for share $j=0$)
		$G_{1,C_i^{(j)}}$	line 9 (and line 11 for share $j=0$)
G_1	Step 1	G_{1,B_i}	line 14
		G_{1,C_i}	line 17
G_2	Step 2	G_{2,B_i}	line 20
		G_{2,C_i}	line 23
G_3	Step 3	G_{3,E_j}	line 27-29 (for one specific j)
G_4	Step 4	G_4	line 32

Starting with the last gadget we will work our way back through the algorithm. We will denote the number of intermediate values probed by the adversary in each gadget with t_0 to t_4, for G_0 to G_4 respectively. The adversary can probe at most $t_c = t_0 + t_1 + t_2 + t_3 + t_4 \leq t$ intermediate variables.

Gadget G_4 Gadget G_4 is by definition t-NI. As it has no sensitive output variables, output probes are not relevant. By definition of t-NI, the t_4 intermediate variables can be simulated with at most t_4 input values.

Gadget G_3 There are S instantiations of gadget G_3, each linked to a specific share number j of the output values of G_2. Each gadget G_{3,E_j} outputs one variable by iteratively calculating:

$$\boldsymbol{E}^{(j)} = \sum_i R_i \cdot \boldsymbol{D}_i^{(j)} \bmod p \cdot 2^{s-1}$$

with the R_i random values.

One can simulate the output value for gadget G_{3,E_j} using all $D_0^{(j)}, \cdots D_{(k+1)n-1}^{(j)}$ corresponding to the number of the gadget j. To calculate an intermediate value $\boldsymbol{E}^{(j)}$ at a certain i in the loop, one needs to know all input values $D_0^{(j)}, \cdots, D_i^{(j)}$ corresponding the number of the gadget j. As such, to simulate the distribution of the probes in G_3 and G_4 one needs to probe at most $t_4 + t_3 \leq t$ of the shares of each G_2 and thus at most $t_4 + t_3 \leq t$ of the output variables of each gadget.

Gadget G_2 The gadgets G_2 are t-SNI secure by definition. From above we know that each gadget is output probed in at most $t_4 + t_3$ output variables. There are t_2 intermediate probes divided over the different gadgets G_{2,B_i} and G_{2,C_i}, the number of which we will denote with t_{2,B_i} or t_{2,C_i} respectively. We have that $t_2 = \sum_i t_{2,B_i} + t_{2,C_i}$. In the following we will focus on the gadgets G_{2,B_i}, but the reasoning can be trivially adapted for G_{2,C_i}.

As each gadget is t-SNI secure we know that any O output variables and t_{2,B_i} intermediate variables can be simulated by at most t_{2,B_i} input variables, if $O + t_{2,B_i} \leq t$. From above we know that $O \leq t_3 + t_4$ and $t_{2,B_i} \leq t_2$ and thus $O + t_{2,B_i} \leq t$. This means that each gadget can be simulated with at most t_{2,B_i} input probes and thus G_2 can be simulated with at most t_2 input probes.

Gadget G_1 The gadgets G_1 are also t-SNI secure by definition. Therefore, with a similar reasoning as for gadget G_2, for at most t_2 output probes and t_1 intermediate probes, we have that the gadget G_1 can be simulated with at most t_1 input probes.

Gadget G_0 Each output variable can be trivially simulated using the corresponding input variable as the gadgets in G_0 are simple operations on one share. The intermediate values of the gadgets G_0 can also be simulated by the corresponding input variable. As such we have $|I_B| + |I_C| \leq O + t_0 \leq t_1 + t_0 \leq t$, which proves our theorem.

□

5 Evaluation

After describing our new approach in depth, we now analyze its performance. We also verify its soundness in practice using side-channel measurements.

5.1 Subroutines

Our masked comparison algorithm requires subroutines for A2B and B2A mask conversion, as well as `BooleanEqualityTest`. There are many ways to instantiate these subroutines. In this section, we detail our choices, aimed at maximum performance.

For A2B conversion, we employ the method of [CGV14, Algorithm 4]. As in [SPOG19], we replace the function `Expand` [CGV14, Algorithm 5] with its t-SNI variant, `RefreshXOR` [BBE+18, Algorithm 8]. The core building block of this specific A2B conversion is a Boolean-masked binary adder. In such an adder, every call to an `XOR` (for the sum) or `AND` (for the carry) is replaced by its masked variants `SecXOR` and `SecAND`. This is trivial for `XOR` in a Boolean masking, but requires special treatment for `SecAND`. The Boolean-masked binary adder requires bit-level manipulations, quickly leading to a large computational

blow-up depending on the size of the input (e.g $p \cdot 2^{f_{bits,B}}$ for $\boldsymbol{B}^A$). [CGV14] described a method to avoid bit-level manipulations, but otherwise keep the same complexity. However, we can exploit the fact that we need many conversions in parallel by bit-slicing the implementation. On the 32-bit Cortex-M4, our A2B implementation uses bit-slicing to compute 32 conversions in parallel.

For B2A conversion, we employ the method of [BCZ18]. We note that this conversion is restricted to power-of-two moduli for the arithmetic masking. However, as mentioned in Section 4, we can freely choose the (large) masking modulus to switch to. We choose a power-of-two modulus, specifically to be able to use this efficient B2A conversion method.

After the right-shift on lines 14 and 17, the boolean shares $\boldsymbol{B}^B$ and $\boldsymbol{C}^B$ are of limited bit-width. An ideal B2A conversion for this task is the one of [SPOG19], which converts each bit separately. However, in our experiments, the B2A conversion of [SPOG19] was outperformed by the conversion of [BCZ18][5].

Finally, `BooleanEqualityTest` requires to check that $\sum_{j=0}^{S-1} \boldsymbol{E}^{(j)} = 0$. We first convert this to $\bigoplus \boldsymbol{E}^B = 0$ using a A2B conversion from arithmetic sharing modulus $p \cdot 2^{s-1}$. Then, for every bit i of $\boldsymbol{E}^B$ we have that $\bigoplus \boldsymbol{E}^B[i] = 0$. This can easily be implemented as the masked circuit:

$$\overline{\boldsymbol{E}^B}[0] \quad \texttt{SecAND} \quad \overline{\boldsymbol{E}^B}[1] \quad \texttt{SecAND} \quad \ldots \quad \texttt{SecAND} \quad \overline{\boldsymbol{E}^B}[\log_2(p \cdot 2^{s-1}) - 1] \tag{26}$$

5.2 Performance Evaluation

To measure the performance of our new masked comparison algorithm, we benchmarked our technique on an STM32F407-DISC1 board that features an ARM-Cortex M4F. We compile with -O3, using `arm-none-eabi-gcc` version 9.2.1. We use the same settings as the popular PQM4 benchmarking framework [KRSS], i.e. a 24 MHz system clock and a 48 MHz TRNG clock. We sample all masking randomness from the on-chip TRNG, and we include the sampling cost as well as the total number of requested random bytes into the benchmarks. On the STM32F407, the TRNG can supply 4 random bytes every 40 TRNG clock cycles, which corresponds to 20 cycles for the main system clock.

In table Table 2, we show the cycle counts of our implementation for the two considered schemes, Kyber and Saber, and their main parameter sets ($k = 3$). We further profile our implementation and break down cycle counts and random bytes in terms of the five steps of Algorithm 7. We benchmark our implementations with collision probability $2^{-s} = 2^{-54}$, corresponding to a maximal bitwidth of 64 bits for the internal variables in Steps 2-4. We provide additional implementations with $s = 118$ (maximal bitwidth 128 bits) and $s = 128$ (maximal bitwidth 138 bits) in Appendix A.

Steps 1 and 2, i.e. the A2B and B2A conversions, clearly constitute the main computational bottlenecks, taking up to 95% of the total algorithm execution time. High-performant conversions are therefore critical, and this motivates our choices such as bit-slicing in the previous section. In our employed conversions, the complexity of A2B is quadratic in the number of shares, whereas the complexity of B2A is exponential. This difference is already visible for Saber with 4 shares (3^{rd}-order masking), where B2A becomes the more costly routine.

Our masked comparison algorithm differs between Saber and Kyber only in the preprocessing and the A2B conversion. The cycle count difference is most noticeable for the A2B conversion. In this conversion, we have for Saber that $p = 2^{10}$, $T = 2^4$, $f_{bits,B} = 3$, and $f_{bits,B} = 6$. In other words, we need 13-bit A2B conversions for $\boldsymbol{B}$ and 10-bit A2B conversions for $\boldsymbol{C}$. For Kyber, we have that $p = 2^{10}$, $T = 2^4$, and $f_{bits,\{B,C\}} = 13$. In

[5] In the sequence $\texttt{B2A}(\texttt{A2B}(\cdot) \gg f_{bits})$ it is also possible to only compute B2A conversions for the carry, further limiting the bit-width. Even with this optimization (called A2A conversion in [VBDK+21]) we found that [BCZ18] is the preferred B2A conversion.

Table 2: Performance breakdown of our masked comparison on ARM Cortex-M4 with collision probability $2^{-s} = 2^{-54}$

Scheme	Comparison	CPU [k]cycles			#random bytes		
		1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}
Saber	**Algorithm 7**	**1,510**	**3,363**	**6,543**	**30,764**	**90,264**	**205,104**
	|-Step 0	|-23	|-23	|-26	|-0	|-0	|-0
	|-Step 1	|-978	|-1,926	|-3,015	|-5,888	|-23,552	|-47,104
	|-Step 2	|-315	|-1,157	|-3,179	|-16,384	|-57,344	|-147,456
	|-Step 3	|-167	|-182	|-197	|-8,192	|-8,192	|-8,192
	|-Step 4	|-27	|-73	|-128	|-300	|-1,176	|-2,352
Kyber768	**Algorithm 7**	**2,421**	**5,253**	**9,712**	**35,500**	**111,256**	**251,184**
	|-Step 0	|-251	|-355	|-467	|-0	|-0	|-0
	|-Step 1	|-1,666	|-3,485	|-5,742	|-10,624	|-44,544	|-93,184
	|-Step 2	|-315	|-1,157	|-3,179	|-16,384	|-57,344	|-147,456
	|-Step 3	|-167	|-182	|-197	|-8,192	|-8,192	|-8,192
	|-Step 4	|-27	|-73	|-128	|-300	|-1,176	|-2,352
	[BGR+21]‡	462 (0.2x)	22,017 (4.2x)	72,568 (7.5x)	12,072†	902,126†	2,434,170†

‡ : [BGR+21] does not have a false-positive collision probability †: Full masked decapsulation, rather than only comparison. Masked comparison accounts for 15%/50%/63% of masked decapsulation cycle counts.

other words, we need 23-bit and 17-bit A2B conversions. Furthermore, to avoid a rounding error as explained in [FVBR+21], for Kyber f_{bits} has to increase logarithmatically with the number of shares, e.g. $f_{bits} = 14$ for 3 shares and $f_{bits} = 15$ for 4 shares. The complexity of A2B is linear in the number of considered bits, leading the increased runtime for Kyber in Step 1. The randomness consumption follows the same trend as the cycle counts. Due to the increased number of iterations within the A2B routine, Kyber requires additional random bytes in Step 1. For B2A, similarly to its cycle counts, the randomness consumption increases exponentially with the number of shares.

We also compare our new technique to results reported by Bos et al. [BGR+21] in their masked implementation of Kyber. Rather than develop a masked compression method for the re-encrypted ciphertext, `DecompressedComparison` chooses to decompress the input ciphertext. Subsequently, a masked range check is employed to determine ciphertext equality. Our method achieves factors 4.2x and 7.5x cycle count improvements for 2^{nd} and 3^{rd}-order maskings, respectively. In their higher-order masked Kyber implementations, the masked comparison accounts for 50%, resp. 63%, of the total execution of masked decapsulation, and our speedup could therefore contribute significantly to reduce overall cycle counts.

In this work, we are concerned with a higher-order masked comparison method. Compared to [BGR+21], for a 1^{st}-order masking we did not employ custom A2B and B2A algorithms, specialized only for this case. As a result, for only 2 shares our algorithm is outperformed by other solutions (factor 5.0x for [BGR+21]).

5.3 Leakage Evaluation

We now present the results of our leakage evaluation. We performed side-channel measurements (power consumption) using the ChipWhisperer Lite [Newa] board. The target device is an STM32F303 board with an ARM Cortex-M4 core running at 7.37 MHz. We capture the traces with a sample rate of 29 MS/s; the sample clock is synchronized to the device clock [Newa]. The code for the side-channel evaluation was compiled using `arm-none-eabi-gcc`, version `9.2.1`. We take measurements on the complete algorithm comparing two polynomials modulo 2^{13} (Saber) with 32 coefficients, the smallest possible option due to the bit-sliced A2B implementation. For the sake of clarity, only these results are included in this section. To ensure that we do not miss any leakage due to the limited sample buffer of the ChipWhisperer Lite, we perform measurements on the smaller building blocks individually and include them in Appendix B.

We show that our method is applicable in practice and does not have any obvious

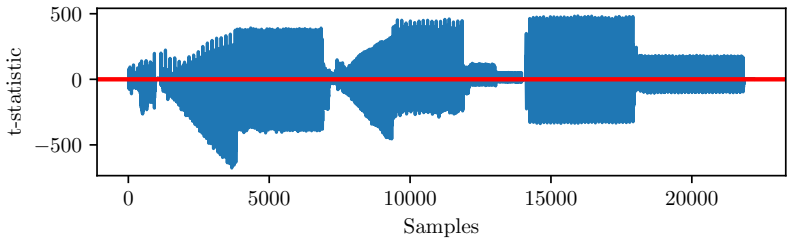
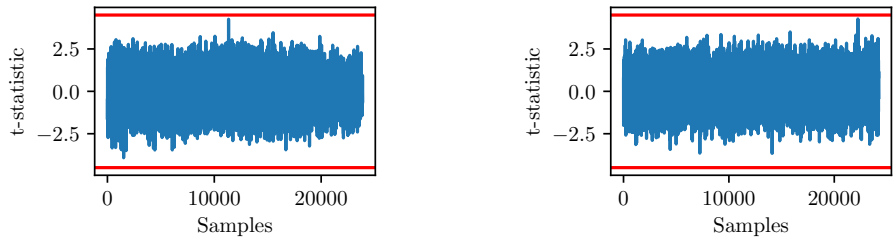


Figure 3: First-order t-test results of the first-order implementation with masks disabled and 10 000 measurements. The red lines indicate the threshold of ±4.5.



(a) t-test for the two-share version.

(b) t-test for the three-share version.

Figure 4: First-order t-test results of the first and second-order implementations with masks enabled and 100 000 measurements. The red lines indicate the threshold of ±4.5.

weaknesses when confronted with first and second-order side-channel attacks. We use the non-specific t-test of the Leakage Assessment Methodology presented in [SM15]. More concretely, we use the fixed + noise variant presented in [BDH+21], which additionally detects leakage caused by possibly unmasked partial comparisons. The first-order t-test statistic is calculated as

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}, \tag{27}$$

where m_0 denotes the sample mean, s_0 denotes the sample variance, and n_0 the sample size for the traces with fixed + noise input. The sample mean, sample variance, and sample size for the random set are denoted by m_1, s_1, and n_1, respectively.

The methodology presented in [SM15] calculates a threshold t-value of 4.5 to obtain a confidence of 0.99999 to correctly reject the null hypothesis, i.e. the two sets are not distinguishable. An absolute t-statistic larger than 4.5 thus indicates leakage.

When the Random Number Generator (RNG) is turned off, as in Figure 3, we can observe that the algorithm shows first-order leakage after only 10 000 measurements. This expected result confirms a correct setup of our measurement equipment.

We activate the RNG in our next experiment using two shares. The results are shown in Figure 4a. We cannot identify any peaks in our measurements with 100 000 executions. Additionally, we provide the first-order t-test statistics for the three-share option of our algorithm in Figure 4b, where we also cannot detect any first-order leakage.

For the bivariate second-order t-test, we first combine each trace at two points in time $\mathcal{T} = \{i, j\}$ using the so-called centered product

$$\prod_{i \in \mathcal{T}} (t^{(i)} - \mu_y^{(i)}) \tag{28}$$

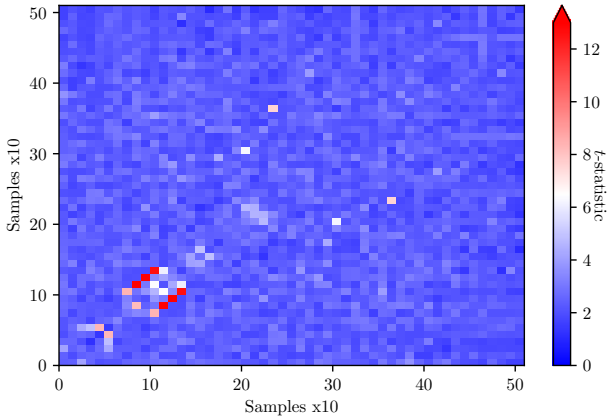


Figure 5: Bivariate second-order t-test for the two-share version with 10 000 traces. The t-statistic shows the absolute value.

with $y \in \{0, 1\}$. Then a second-order t-test is performed on the resulting two-dimensional traces. As presented in [SM15], d-th order central moments $CM_d = E((X - \mu)^d)$ are required. For our evaluation, the mean of the first-order t-test μ is replaced with the second-order central moment CM_2, whereas the variance is set to $CM_4 - CM_2^2$. Using the methodology of [SM15], we calculate the t-statistic iteratively without separate sampling, combination, and calculation steps. However, as this step is computationally very expensive, we have to reduce the number of sample points during capturing significantly. This carries the risk of missing leakage and thus a more efficient higher-order evaluation approach might be interesting future work.

In contrast to the first-order case, the effects of applying many t-tests simultaneously are non-negligible in the second-order scenario. Similar to [BPO+20], we can apply the Šidák Correction $t_{th} = Q_t(1 - \sqrt[L]{1-\alpha}, \nu)$ proposed in [BPG18], where Q_t is the quantile function of the t-distribution, L is the trace length, α is the confidence level, and ν is the degree of freedom, to obtain a valid threshold value. In our example of 500^2 sample points, $\alpha = 0.00001$ and 100 000 traces, the threshold t-value results in 6.50.

In the first-order implementation with two shares, the bivariate second-order t-test shows some clear leakage points even when only 10 000 captured traces are taken into account. This is expected behavior because a first-order implementation, in general, can not withstand second-order attacks. We graphically illustrate the result in Figure 5, where we depict the color of each 10x10 time sample square according to the maximum absolute t-statistic.

In contrary to the first-order implementation, a second-order implementation should not leak using a second-order t-test. The result of the second-order t-test of our three-share version is shown in Figure 6. As expected, even with 100 000 traces, no leakage points above 6.5 appear. Thus, all practical experiments confirm our theoretical results.

6 Future Work

Our new masked comparison algorithm heavily draws on A2B and B2A conversion techniques. Consequently, its performance depends crucially on the performance of these two algorithms. Table-based conversions are especially appealing, but they have been restricted to first-order masked implementations [CT03, Deb12, VDV21]. Concurrently with our work, higher-order table-based mask conversion methods have been proposed [CGMZ21],

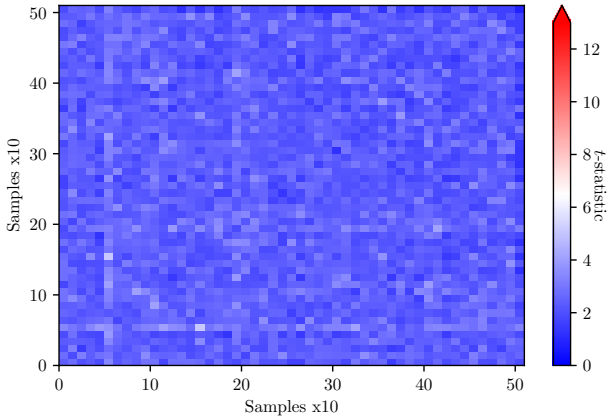


Figure 6: Bivariate second-order t-test for the three-share version with 100 000 traces. The t-statistic shows the absolute value.

specifically focused on lattice-based cryptography. These methods boast increased performance compared to the A2B and B2A conversions we use in this work. The authors use their new techniques to mask the CPA-secure decryption and the binomial sampling, but specifically leave the masking of the polynomial comparison as future work. Opportunely, the novel masked comparison method that we described in this work is generic and can work with any A2B or B2A. Therefore, integrating these new higher-order table-based conversion methods into our masked comparison is a clear direction for future work.

Acknowledgments

We would like to thank Nigel Smart and Barry van Leeuwen for the interesting discussions on the effect of prime and power-of-two moduli on the false positive probability and for pointing us to the relevant proving techniques in MPC literature. We would also like to thank Thomas Pöppelmann for the discussions on how to optimize the masked comparison operation and Julius Hermelink for his support on integrating his key-recovery algorithm in our attack on the hash-based approach.

This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, the Research Council KU Leuven (C16/15/058), the Horizon 2020 ERC Advanced Grant (695305 Cathedral) and SRC grant 2909.001. Michiel Van Beirendonck is funded by an FWO PhD fellowship strategic basic research. Jan-Pieter D'Anvers is funded by FWO (Research Foundation – Flanders) as junior post-doctoral fellow (contract number 133185 / 1238822N LV). This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project "Aquorypt" (16KIS1017). Presented project results were partly supported by the project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927. The authors would like to thank the Chair for Communication Systems and Network Security as well as the research institute CODE at the Bundeswehr University in Munich, headed by Prof. Dreo, for their comments and improvements.

References

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based

signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. Improved high-order conversion from Boolean to arithmetic masking. *IACR TCHES*, 2018(2):22–45, 2018. https://tches.iacr.org/index.php/TCHES/article/view/873.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison. *IACR TCHES*, 2021(3):334–359, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8977.

[BDK+18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 353–367, 2018.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9064.

[BPG18] Florian Bache, Christina Plump, and Tim Güneysu. Confident leakage assessment - A side-channel evaluation framework based on confidence intervals. In Jan Madsen and Ayse K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 1117–1122. IEEE, 2018.

[BPO+20] Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based kems. *IACR TCHES*, 2020(3):483–507, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8598.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 769–798. Springer, Heidelberg, August 2018.

[CDH+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[CGMZ21] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order table-based conversion algorithms and masking lattice-based encryption. Cryptology ePrint Archive, Report 2021/1314, 2021. https://ia.cr/2021/1314.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Heidelberg, September 2014.

[CT03] Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to Boolean masking. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 89–97. Springer, Heidelberg, September 2003.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020.

[Deb12] Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to Boolean masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 107–121. Springer, Heidelberg, September 2012.

[DGJ+19] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 565–598. Springer, Heidelberg, April 2019.

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Heidelberg, May 2018.

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum schemes. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, TIS'19, page 2–9, New York, NY, USA, 2019. Association for Computing Machinery.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

[FVBR+21] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. Cryptology ePrint Archive, Report 2021/479, 2021. https://eprint.iacr.org/2021/479.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on kyber. Cryptology ePrint Archive, Report 2021/1222, 2021. https://eprint.iacr.org/2021/1222.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, Heidelberg, August 2010.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR TCHES*, 2021(4):676–707, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9079.

[Newa] NewAE. CW1173 ChipWhisperer-Lite. https://rtfm.newae.com/Capture/ChipWhisperer-Lite.html.

[Newb] NewAE. CW308T-STM32F. https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F.html.

[NIS16] NIST Computer Security Division. Post-Quantum Cryptography Standardization, 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. https://tches.iacr.org/index.php/TCHES/article/view/836.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR TCHES*, 2021(2):37–60, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8787.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8592.

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, Heidelberg, September 2015.

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 534–564. Springer, Heidelberg, April 2019.

[UXT+21] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. Cryptology ePrint Archive, Report 2021/849, 2021. https://ia.cr/2021/849.

[VBDK+21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM JETC*, 17(2):10:1–10:26, 2021.

[VDV21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, and Ingrid Verbauwhede. Analysis and comparison of table-based arithmetic to boolean masking. *IACR TCHES*, 2021(3):275–297, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8975.

[XPRO20] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, and David Oswald. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. Cryptology ePrint Archive, Report 2020/912, 2020. https://eprint.iacr.org/2020/912.

[YJ00] 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.

A Supplementary Security

Our main benchmarks use $s = 54$, corresponding to a collision probability of 2^{-54}, and requiring a 64-bit type in Steps 2-4 of our Algorithm 7. We believe this collision probability will be sufficiently low for the majority of use cases, but do provide implementation results for $s = 118$ and $s = 128$ in Tables 3 and 4. These implementations require, respectively, a custom 128-bit type and 138-bit type in Steps 2-4 of Algorithm 7. Moving up from a 64-bit type causes bit-wise operations and randomness sampling to increase by roughly a factor of two in these steps, whereas the multiplications in Step 3 incur a higher overhead. Even with a reduced collision probability requiring custom types, our algorithm outperforms [BGR+21].

Table 3: Performance breakdown of our masked comparison on ARM Cortex-M4 with collision probability $2^{-s} = 2^{-118}$

Scheme	Comparison	CPU [k]cycles			#random bytes		
		1st	2nd	3rd	1st	2nd	3rd
Saber	**Algorithm 7**	**2,155**	**5,352**	**11,096**	**55,628**	**156,944**	**363,040**
	|-Step 0	|-23	|-23	|-26	|-0	|-0	|-0
	|-Step 1	|-978	|-1,926	|-3,015	|-5,888	|-23,552	|-47,104
	|-Step 2	|-629	|-2,702	|-7,146	|-32,768	|-114,688	|-294,912
	|-Step 3	|-481	|-568	|-679	|-16,384	|-16,384	|-16,384
	|-Step 4	|-46	|-130	|-230	|-588	|-2,320	|-4640
Kyber768	**Algorithm 7**	**3,069**	**7,241**	**14,262**	**60,364**	**177,936**	**409,120**
	|-Step 0	|-251	|-355	|-467	|-0	|-0	|-0
	|-Step 1	|-1,666	|-3,485	|-5,742	|-10,624	|-44,544	|-93,184
	|-Step 2	|-629	|-2,702	|-7,146	|-32,768	|-114,688	|-294,912
	|-Step 3	|-481	|-568	|-679	|-16,384	|-16,384	|-16,384
	|-Step 4	|-46	|-130	|-230	|-588	|-2,320	|-4,640
	[BGR+21]‡	462 (0.2x)	22,017 (3.0x)	72,568 (5.1x)	12,072†	902,126†	2,434,170†

‡ : [BGR+21] does not have a false-positive collision probability †: Full masked decapsulation, rather than only comparison. Masked comparison accounts for 15%/50%/63% of masked decapsulation cycle counts.

Table 4: Performance breakdown of our masked comparison on ARM Cortex-M4 with collision probability $2^{-s} = 2^{-128}$

Scheme	Comparison	CPU [k]cycles			#random bytes		
		1st	2nd	3rd	1st	2nd	3rd
Saber	**Algorithm 7**	**2,436**	**6,261**	**13,095**	**63,876**	**185,836**	**437,208**
	|-Step 0	|-23	|-23	|-26	|-0	|-0	|-0
	|-Step 1	|-978	|-1,926	|-3,015	|-5,888	|-23,552	|-47,104
	|-Step 2	|-779	|-3,393	|-8,845	|-40,960	|-143,360	|-368,640
	|-Step 3	|-610	|-779	|-965	|-16,384	|-16,384	|-16,384
	|-Step 4	|-48	|-138	|-245	|-644	|-2,540	|-5,080
Kyber768	**Algorithm 7**	**3,350**	**8,150**	**16,261**	**68,612**	**206,828**	**483,288**
	|-Step 0	|-251	|-355	|-467	|-0	|-0	|-0
	|-Step 1	|-1,666	|-3,485	|-5,742	|-10,624	|-44,544	|-93,184
	|-Step 2	|-779	|-3,393	|-8,845	|-40,960	|-143,360	|-368,640
	|-Step 3	|-610	|-779	|-965	|-16,384	|-16,384	|-16,384
	|-Step 4	|-48	|-138	|-245	|-644	|-2,540	|-5,080
	[BGR+21]‡	462 (0.1x)	22,017 (2.7x)	72,568 (4.5x)	12,072†	902,126†	2,434,170†

‡ : [BGR+21] does not have a false-positive collision probability †: Full masked decapsulation, rather than only comparison. Masked comparison accounts for 15%/50%/63% of masked decapsulation cycle counts.

B Supplementary TVLA

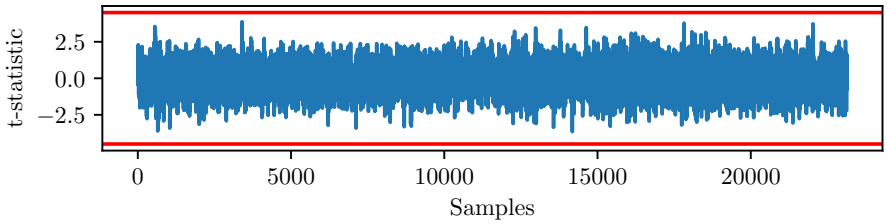


Figure 7: First-order t-test results of the first-order `A2B` implementation with 100 000 measurements. The red lines indicate the threshold of 4.5.

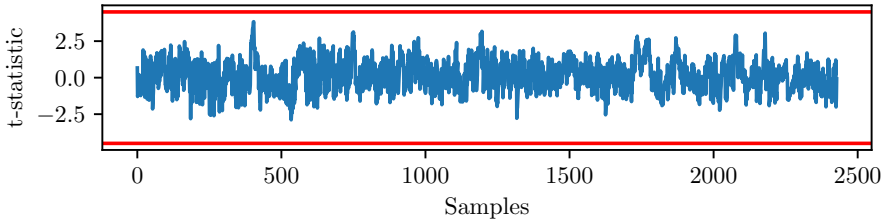


Figure 8: First-order t-test results of the first-order `B2A` implementation with 100 000 measurements. The red lines indicate the threshold of 4.5.

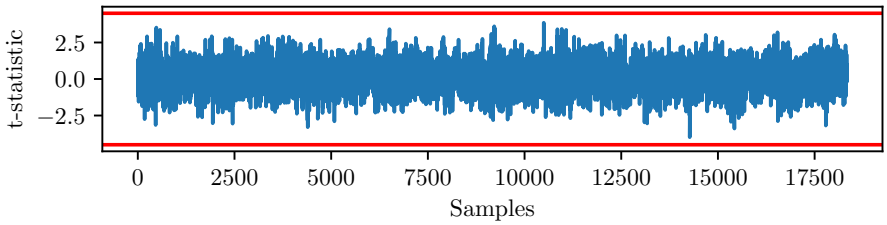


Figure 9: First-order t-test results of the first-order `ReduceComparisons` implementation with 100 000 measurements. The red lines indicate the threshold of 4.5.

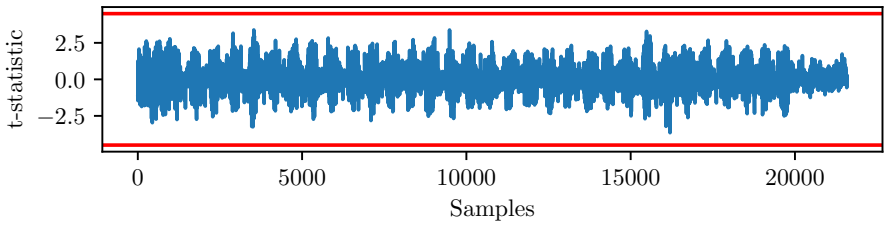


Figure 10: First-order t-test results of the first-order `BooleanEqualityCheck` implementation with 100 000 measurements. The red lines indicate the threshold of 4.5.