IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 3, pp. 334–359. DOI:10.46586/tches.v2021.i3.334-359

Attacking and Defending Masked Polynomial Comparison for Lattice-Based Cryptography

Shivam Bhasin[1], Jan-Pieter D'Anvers[2], Daniel Heinz[34], Thomas Pöppelmann[4], Michiel Van Beirendonck[2]

[1] Temasek Labs, Nanyang Technologival University, Signapore
sbhasin@ntu.edu.sg
[2] imec-COSIC KU Leuven, Kasteelpark Arenberg 10 - bus 2452, 3001 Leuven, Belgium
{firstname}.{lastname}@esat.kuleuven.be
[3] Research Institute CODE, Universität der Bundeswehr München, Germany
daniel.heinz@unibw.de
[4] Infineon Technologies, Am Campeon 1-15, 85579 Neubiberg, Germany
thomas.poeppelmann@infineon.com

Abstract. In this work, we are concerned with the hardening of post-quantum key encapsulation mechanisms (KEM) against side-channel attacks, with a focus on the comparison operation required for the Fujisaki-Okamoto (FO) transform. We identify critical vulnerabilities in two proposals for masked comparison and successfully attack the masked comparison algorithms from TCHES 2018 and TCHES 2020. To do so, we use first-order side-channel attacks and show that the advertised security properties do not hold. Additionally, we break the higher-order secured masked comparison from TCHES 2020 using a collision attack, which does not require side-channel information. To enable implementers to spot such flaws in the implementation or underlying algorithms, we propose a framework that is designed to test the re-encryption step of the FO transform for information leakage. Our framework relies on a specifically parametrized t-test and would have identified the previously mentioned flaws in the masked comparison. Our framework can be used to test both the comparison itself and the full decapsulation implementation.

Keywords: Lattice-Based Cryptography · Side-Channel Attack · Fujisaki-Okamoto transform

1 Introduction

Conventional public-key cryptography, like RSA and ECC, suffers from an ever-increasing threat as large-scale quantum computers advance closer to reality. As a consequence, a public effort to standardize post-quantum cryptography (PQC) was initiated by NIST in 2017 [NIS16]. With the NIST process recently reaching the third round, implementation security and protection against side-channel or fault attacks is emerging as an important criterion for standardization [AASA+20].

So far, several works have shed light on the side-channel vulnerabilities of lattice-based PQC schemes. Authors exploit vulnerabilities at different levels, including but not limited to, building blocks like the number theoretic transform (NTT) [PPM17], message encoding [ACLZ20], or the Fujisaki-Okamoto (FO) transform [DTVV19, GJN20, RRCB20]. A commonly applied countermeasure to protect implementations against side-channel attacks is to mask every sensitive step of the algorithm. An important observation is that lattice-based key encapsulation mechanisms (KEM) achieve semantic security with respect to chosen-ciphertext

Received: 2020-10-15 Accepted: 2020-12-15 Published: 2021-07-09

attacks (CCA) by application of the FO transform [FO99, HHK17]. Even though the additional operations in this transform do not directly process secret information, they still have to be protected against leakage as the processed data depends on the secret key. The importance of this aspect has recently been demonstrated by side-channel attacks [DTVV19, RRCB20] on lattice-based schemes and more recently a timing attack [GJN20] on FrodoKEM [NAB+20].

Besides reports of first-order masked implementations of a scheme similar to NewHope in [OSPG18] and Saber in [BDK+20], the need to protect the FO transform has also led to works that specifically look into building blocks and also their higher-order security. While arithmetic masking to protect arithmetic operations in the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^n+1)$ can be scaled to higher orders easily due to the linearity of the operation, it is not trivial to achieve this property for polynomial sampling [SPOG19] or the final comparison step[1] [BPO+20] of the FO transform.

In short, securing the FO transform and the required hashing, re-encryption and comparison is non-trivial. And while works like [OSPG18, BPO+20] provide a security proof, it is still important to check the correctness of their assumptions and arguments. This aspect is reinforced by the fact that small leakages or inconsistencies in the security reasoning can already be devastating. In many scenarios (e.g. [DTVV19, RRCB20, GJN20]) an attacker can retrieve the full secret key by any information on whether data processed by the FO transform differs between different input ciphertexts.

Contributions

In this work, we focus on flaws in previously proposed masked comparisons that were used as part of the FO transformation. We show that the approach for a first-order masked comparison from [OSPG18] leads to a vulnerable implementation. Moreover, we describe two attacks on the higher-order masked comparison from [BPO+20]. One is a collision attack that does not require any side-channel information. The second attack breaks the scheme by a first-order side-channel attack. We show that all these attacks allow an adversary to efficiently retrieve the secret key by applying it on a Kyber implementation that uses the aforementioned vulnerable masked comparisons.

As a result, we indicate that the masked comparison should be an atomic operation applied to all inputs, such that no comparison result on a subset of coefficients is leaked. We then fix [OSPG18] using the strategy of [BDK+20], and we propose a correction of the flaw in the [BPO+20] scheme.

We also highlight an issue in the leakage testing done by the original authors [OSPG18, BPO+20], namely the existence of non-malevolent output leakage. This non-malevolent leakage can obfuscate real leakage, making it harder to capture real vulnerabilities as for example the ones reported in this paper. In response, we introduce a framework that is explicitly designed for leakage testing in security-critical parts of the FO transform. To validate our approach, we show that our framework makes it easier to catch the problems present in [OSPG18] and [BPO+20].

2 Preliminaries

In this section, we introduce our notation and provide details on the Kyber KEM. In addition, we recall previous work on masked implementations of lattice-based KEMs and the protection of the comparison operation in the FO re-encryption against side-channel attacks.

[1] Note that the term 'equality check' might be more appropriate here. However, we choose to stick to the terminology used in literature.

2.1 Notation

For $x \in \mathbb{R}$, we write $\lceil x \rfloor$ to mean the closest integer to x (where $\lceil y+\frac{1}{2} \rfloor := y+1$ for $y \in \mathbb{Z}$). For $a,b \in \mathbb{Z}$, we write $a \bmod^{(+)} b$ for the unique integer $\hat{a} \equiv a \bmod b$ such that $0 \leq \hat{a} < b$, and similarly $a \bmod^{(\pm)} b$ denotes the integer $\hat{a} \equiv a \bmod b$ such that $-b/2 \leq \hat{a} < b/2$. We extend these definitions to tuples, vectors, matrices, and polynomials a over $\mathbb{Z}$ component-wise. When the exact representation is not important, we simply write $a \bmod b$. Let $\mathbb{Z}_q$ denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$ for an integer $q \geq 1$. Let $R = \mathbb{Z}[X]/(f)$, where usually $f = X^n+1$ for n a power of 2, and $R_q = R/(q) = \mathbb{Z}_q[X]/(f)$ for some positive integer q. We let R^k (resp. R_q^k) be a ring module of dimension k over R (resp. R_q). We identify equivalence classes in R_q with their representative polynomial with coefficients $\bmod^{(+)} q$. By $\{0,1\}^z$ we denote the set of z bits and by $\{0,1\}^*$ we denote the set of bits of arbitrary length. We use the notation a_i or $a[i]$ for $i = 0,\ldots,n-1$ to access the i-th coefficient of a. Matrices of elements in R_q are denoted as upper case letters.

For a given set S and a probability distribution D over S, we use $s \xleftarrow{r} D$ to mean $s \in S$ sampled according to D using coins r. In addition, we use $s \xleftarrow{\$} S$ to mean $s \in S$ sampled uniformly at random from S. Hereby, $U(q)$ denotes the uniform distribution on R_q, whereas χ denotes an error distribution to be defined for the specific algorithm.

If a variable v gets overwritten as part of a loop (e.g. $v \leftarrow v/2$), we may refer to the variable v after step i of the loop as $v^{[i]}$ (e.g. $v^{[i]} \leftarrow v^{[i-1]}/2$). A plain variable A that is split into integer S shares is denoted in bold notation as $\boldsymbol{A}$. The plain value of an arithmetically shared $\boldsymbol{A} \bmod q$ (in $\mathbb{Z}_q$,R_q, or R_q^k) is reconstructed as $A = \sum_{j=1}^{S} \boldsymbol{A}^{(j)} \bmod q$. To make clear that we refer to shares we use round brackets in the $\boldsymbol{A}^{(j)}$ notation to refer to the j-th share for $1 \leq j \leq S$. We also use the notation $\boldsymbol{A} \in \mathbb{Z}_q^{(S)}$ to denote that $\boldsymbol{A}$ is split into S shares and each share is in $\mathbb{Z}_q$. Individual shares, e.g. $\boldsymbol{A}^{(1)}$, are denoted in bold to make it easier to identify them in algorithms. By $||$ we denote the concatenation operation. The input to a comparison algorithm COMPARE is usually denoted as A and $\tilde{A}$ and may be in $\mathbb{Z}_q$,R_q,$\{0,1\}^*$ depending on the context. The result of $\text{COMPARE}(A,\tilde{A})$ is either $true$ if $A = \tilde{A}$ and $false$ otherwise.

2.2 The Kyber Key Encapsulation Mechanism

In this work, we use the Kyber KEM to showcase our attacks and countermeasures. Kyber is an IND-CCA2-secured KEM and a finalist in round three of the NIST PQC standardization process [SAB+20]. Kyber was first described in [BDK+18] and uses the Fujisaki-Okamoto (FO) transformation [FO99, HHK17]. The FO transform is applied to an intermediate chosen-plaintext attack (CPA) secured public-key encryption (PKE) to achieve IND-CCA2 security. The three parameter sets Kyber512, Kyber768, and Kyber1024 are claimed to have a security level equivalent to the security of AES-128, AES-192, and AES-256, respectively. All Kyber variants share the parameters $n = 256$, $q = 3329$, $\eta_2 = 2$ and the security level is defined by appropriately setting k, η_1, d_t, d_u, and d_v. Computations in Kyber are performed in the ring $R_q = \mathbb{Z}_q[X]/(X^n+1)$ and by $\mathcal{M} = \{0,1\}^n$ we denote the plaintext space, where each message $m \in \mathcal{M}$ can be seen as a polynomial in R_q with coefficients in $\{0,1\}$. Kyber uses ciphertext compression to reduce the size of the ciphertext following standard techniques also applied by other lattice-based schemes. The Kyber compression is defined as:

$$\text{COMPRESS}_q(x,d) := \lceil (2^d/q) \cdot x \rfloor \bmod^{(+)} 2^d, \tag{1}$$

$$\text{DECOMPRESS}_q(x,d) := \lceil (q/2^d) \cdot x \rfloor. \tag{2}$$

For later reference, we provide a simplified[2] version of the public-key encryption scheme KYBER.CPA = (KYBER.CPA.GEN, KYBER.CPA.ENC, KYBER.CPA.DEC) as in Algorithms 1, 2, and 3. We set χ_η as the centered binomial distribution with support $\{-\eta,\ldots,\eta\}$,

[2] We use a more straightforward description of the sampling procedures and encoding into byte arrays. To avoid a notation clash on the use of r we renamed the MLWE secrets to s_g in key generation and s_e in encapsulation.

Algorithm 1: KYBER.CPA.GEN.

1 $(\rho,\sigma) \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$;
2 $\hat{A} \xleftarrow{\rho} U(q)^{k \times k}$;
3 $(s_g,e) \xleftarrow{\sigma} \chi^k_{n,\eta_1} \times \chi^k_{n,\eta_1}$;
4 $\hat{s}_g \leftarrow \text{NTT}(s_g)$;
5 $\hat{e} \leftarrow \text{NTT}(e)$;
6 $\hat{t} \leftarrow \hat{A} \circ \hat{s}_g + \hat{e}$;
7 **return** $pk_{CPA} := (\hat{t},\rho), sk_{CPA} := \hat{s}_g$;

Algorithm 2: KYBER.CPA.ENC.

Input: $pk_{CPA} = (\hat{t},\rho)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
1 $\hat{A} \xleftarrow{\rho} U(q)^{k \times k}$;
2 $(s_e,e_1,e_2) \xleftarrow{r} \chi^k_{n,\eta_1} \times \chi^k_{n,\eta_2} \times \chi_{n,\eta_2}$;
3 $\hat{s}_e \leftarrow \text{NTT}(s_e)$;
4 $u \leftarrow \text{INTT}(\hat{A} \circ \hat{s}_e) + e_1$;
5 $v \leftarrow \text{INTT}(\hat{t} \circ \hat{s}_e) + e_2 + \lceil \frac{q}{2} \rfloor \cdot m)$;
6 $c_1 \leftarrow \text{COMPRESS}_q(u,d_u)$;
7 $c_2 \leftarrow \text{COMPRESS}_q(v,d_v)$;
8 **return** $c := (c_1,c_2)$;

Algorithm 3: KYBER.CPA.DEC.

Input: $sk_{CPA} = \hat{s}_g$
Input: $c = (u,v)$
1 $u \leftarrow \text{DECOMPRESS}_q(u,d_u)$;
2 $v \leftarrow \text{DECOMPRESS}_q(v,d_v)$;
3 $m = \text{COMPRESS}_q(v - \text{INTT}(\hat{s}_g^T \circ \text{NTT}(u)),1)$;
4 **return** m ;

and let $\chi_{n,\eta}$ be the distribution of polynomials of degree n with entries independently sampled from χ_η. When we apply the NTT to a vector of polynomials, the NTT gets applied to each polynomial individually.

2.3 The Fujisaki-Okamoto Transform and Physical Attacks

The FO transformation [FO99, HHK17] of Kyber requires, besides access to Algorithms 1, 2, and 3, two different hash functions H and G as well as a key derivation function (KDF). For future reference, we now provide a simplified[3] description of the IND-CCA2 Kyber KEM in Algorithms 4, 5, 6. The main idea of the conversion is to check the validity of a ciphertext in KYBER.CCAKEM.DECAPS after decryption by performing a so-called re-encryption. In Algorithm 6, a candidate ciphertext $c' :=$ KYBER.CPA.ENC(pk,m',r') (see Line 4) is obtained and then compared with the input ciphertext c. The goal is to detect maliciously crafted ciphertexts that could be used to reveal the secret key. In Figure 1, an overview of the application of the FO transformation on Kyber KEM is given, where operations that depend on the secret key are indicated in grey.

The security properties of the FO transform only hold if an adversary is not able to learn information on intermediate values processed during the re-encryption operation. This is because the input to the re-encryption depends on the decryption which uses the secret key. The only information that an attacker should get is an accept or a failure. This information can either be provided explicitly as a reject/fail flag or implicitly by outputting either the correct key or in case of a failure a random/constant key. Thus, side-channel leakage in any form from the FO transform could lead to serious security flaws.

D'Anvers et al. [DTVV19] first presented an implementation attack, using timing information present in the error correction of LAC and Ramstake. Timing leakage was also exploited in [GJN20], this time in the FO validation check of FrodoKEM. Even constant-time implementations can still be vulnerable to other types of side-channel leakage. Electromagnetic radiation from a hashing step in the FO transform has similarly been exploited as a

[3] We removed details on the encoding of the key and ciphertext into byte arrays.

Algorithm 4:
KYBER.CCAKEM.GEN.

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

Algorithm 5:
KYBER.CCAKEM.ENCAPS.

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

Algorithm 6:
KYBER.CCAKEM.DECAPS.

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

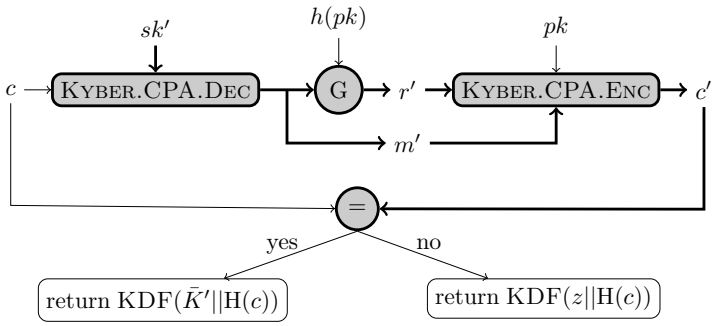


Figure 1: Decapsulation of Kyber. All operations in grey are influenced by the long term secret sk.

covert channel in [RRCB20], highlighting the need for side-channel countermeasures on top of constant-time implementation.

In this paper, we specifically look at the protection of the comparison operation. We will summarize the two masked comparison operations analyzed in this paper below.

2.4 Masked Comparison from [OSPG18]

To goal of [OSPG18] is to compare a public polynomial $\tilde{A}$ with a sensitive and first-order masked polynomial A, which is split in two shares $\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)}$ so that $A = \boldsymbol{A}^{(1)} + \boldsymbol{A}^{(2)}$. The main idea of the OSPG method is to introduce an additional hashing step before the comparison.

Variables $A,\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)},\tilde{A}$ can be defined as polynomials in R_q or just coefficients of a vector in $\mathbb{Z}_q^n$ as the ring structure of R_q is not relevant and the only required operation is coefficient-wise addition mod q. The main idea is that the hashing step prevents leakage of the sensitive polynomial A. Thus, the comparison is rewritten as

$$\boldsymbol{A}^{(1)} + \boldsymbol{A}^{(2)} == \tilde{A} \tag{3}$$

$$\Leftrightarrow \quad \boldsymbol{A}^{(1)} == \tilde{A} - \boldsymbol{A}^{(2)} \tag{4}$$

$$\Leftrightarrow \quad \text{H}(\boldsymbol{A}^{(1)}) == \text{H}(\tilde{A} - \boldsymbol{A}^{(2)}), \tag{5}$$

where the last step relies on the collision-resistance of a cryptographic hash function $\mathrm{H} : R_q \to \{0,1\}^d$ with d output bits (e.g., $d = 256$). The attacker does not learn additional information on A in case of a failed comparison as the unmasking is done on a hashed value. This approach is illustrated in Algorithm 7.

Algorithm 7: Masked Comparison of sensitive A split into shares $\boldsymbol{A}^{(1)}$, $\boldsymbol{A}^{(2)}$ with public $\tilde{A}$ [OSPG18, Algorithm 5]

Input : $\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)} \in R_q$ such that $\boldsymbol{A}^{(1)} + \boldsymbol{A}^{(2)} = A \bmod q$, $\tilde{A} \in R_q$
Output : $\tilde{A} \stackrel{?}{=} A$
1 $\boldsymbol{A}^{(1)} \leftarrow \tilde{A} - \boldsymbol{A}^{(1)}$
2 $\boldsymbol{D}^{(1)} \leftarrow \mathrm{H}(\boldsymbol{A}^{(1)})$
3 $\boldsymbol{D}^{(2)} \leftarrow \mathrm{H}(\boldsymbol{A}^{(2)})$
4 **return** $(\boldsymbol{D}^{(1)} \oplus \boldsymbol{D}^{(2)}) \stackrel{?}{=} 0$

In [OSPG18], there are three input ciphertext components, $\tilde{c}_1$, $\tilde{c}_2$, and $\tilde{c}_4$ that must be compared to their re-encrypted counterparts c_1, c_2, and c_4 The authors mention that the adversary can adaptively change c_2 or c_4 and use the output leakage of $\textsc{compare}(c_2, \tilde{c}_2)$ (resp. $\textsc{compare}(c_4, \tilde{c}_4)$) to distinguish the decrypted m_{cpa}. They note that the same does not apply to the input component $\tilde{c}_1$, and propose to first execute $\textsc{compare}(c_1, \tilde{c}_1)$ individually. Only if this final comparison is valid, the other two comparisons may be conducted [OSPG18, Section 3.4]. Note that we show in Section 3.2 that this assumption is incorrect and that even the output of $\textsc{compare}(c_1, \tilde{c}_1)$ leaks sensitive information.

2.4.1 Application to Saber

For their implementation of Saber [BDK+20], the authors focus on first-order security and therefore re-use the masked comparison of [OSPG18]. In Saber, it is required to check two sensitive and shared re-encrypted ciphertext components $\boldsymbol{c_1} \in R_p^{(2)}$ and $\boldsymbol{c_2} \in R_t^{(2)}$ for equality with the user-provided ciphertext $\tilde{c}_1 \in R_p$ and $\tilde{c}_2 \in R_t$. Different from the original method, the authors implement only a single check. In order to do so, they instantiate the hash function as $\mathrm{H} : R_p \times R_t \to \{0,1\}^d$ with a simple concatenation of the inputs. In other words, it is checked whether

$$\mathrm{H}(\boldsymbol{c}_1^{(1)} \| \boldsymbol{c}_2^{(1)}) \stackrel{?}{=} \mathrm{H}((\tilde{c}_1 - \boldsymbol{c}_1^{(2)}) \| (\tilde{c}_2 - \boldsymbol{c}_2^{(2)})). \tag{6}$$

2.5 Masked Comparison from [BPO+20]

A method to achieve higher-order security for the masked comparison was presented by Bache, Paglialonga, Oder, Schneider, and Güneysu in [BPO+20]. They took a significantly different approach, as it is unclear how the hashing-based approach from [OSPG18] can be used with a higher-order masking scheme. Moreover, a straightforward solution based on a higher-order protected A2B conversion would seem to be too inefficient.

For consistency with related algorithms and our introduction of Kyber, we change the notation used in [BPO+20] when recalling the BPO approach. By n' we denote the number of input coefficients to the comparison algorithm. As the algorithm does not exploit any ring structure, the number of input coefficients n' (originally k) may be $n' = n$ for a polynomial in R_q but could also be the concatenated vector representation of various polynomials in R_q (e.g., $n' = kn$ to account for the modular structure in Kyber). Inputs can be in $\mathbb{Z}_q$ (originally $\mathbb{F}_q$) for a prime q. A prime modulus is required by the security proof and thus the approach would not be directly applicable to Kyber, due to compression, or Saber, due to the

Algorithm 8: `MaskedSum` of m-th set according to [BPO+20, Algorithm 5]

Input : An m-th subset $\boldsymbol{A}$ consisting of shared coefficients $\boldsymbol{A}_1,\ldots,\boldsymbol{A}_l \in \mathbb{Z}_q^{(S)}$ such that $\sum_{j=1}^{S} \boldsymbol{A}_i^{(j)} = A_i \bmod q$, An m-th subset $\tilde{A}$ consisting of coefficients $\tilde{A}_1,\ldots,\tilde{A}_l \in \mathbb{Z}_q$

Output : $B_m, \tilde{B}_m \in \mathbb{Z}_q$

$(\boldsymbol{B}_m^{(i)})_{1\le i\le n} \leftarrow 0$
$B_m \leftarrow 0$
$\tilde{B}_m \leftarrow 0$
for $i = 1$ **to** l **do**
  $R_1 \xleftarrow{\$} \mathbb{Z}_q$
  $R_2 \xleftarrow{\$} \mathbb{Z}_q$
  for $j = 1$ **to** S **do**
    $B_m^{(j)} \leftarrow B_m^{(j)} + (\boldsymbol{A}_i^{(j)} + R_1)\cdot R_2 \bmod q$
  end
  $\tilde{B}_m \leftarrow \tilde{B}_m + (\tilde{A}_i + n\cdot R_1)\cdot R_2 \bmod q$
end
for $j = 1$ **to** S **do**
  $B_m \leftarrow B_m + B_m^{(j)} \bmod q$
end
return $B_m, \tilde{B}_m$

power-of-two modulus. The comparison algorithm requires the sensitive input A to be shared in S (originally n) shares. The shares of A are compared with the unshared public value $\tilde{A}$.

The idea of [BPO+20] is to perform this comparison on summed up and randomized subsets of $\boldsymbol{A}$ and $\tilde{A}$. The value $\boldsymbol{A}$ is assumed to be a vector of n' coefficients in $\mathbb{Z}_q^{(S)}$ and $\tilde{A}$ is assumed to be a vector of n' coefficients in $\mathbb{Z}_q$. The actual value of n' depends on the scheme in which the masked comparison is instantiated, e.g., $n' = (k+1)n$ for Kyber. To run the comparison, both are split into x sets of cardinality $l = \frac{z}{x}$ (originally, k is used instead of z) assuming that l divides n'. Then Algorithm 8 is run on each m-th subset and for each result $\textsc{compare}(B_m, \tilde{B}_m)$ for $0 \le m < x$ is run. If all comparisons return *true*, the final result is *true*. Note that we follow the notation of [BPO+20] to avoid a notation clash by denoting the m-th subsets $\boldsymbol{A}_m$ and $\tilde{A}_m$ just as $\boldsymbol{A}$ and $\tilde{A}$ in Algorithm 8 to be able to use the subscript for accessing individual coefficients[4].

3 Attack

In this section, we present three attacks against masked comparison algorithms, one against the comparison proposed in [OSPG18] and two against [BPO+20]. Our attacks make use of the fact that it is possible to submit slightly modified ciphertexts and observe whether a decryption failure occurs. These attacks invalidate the implicit assumption of [OSPG18] and [BPO+20] that the output of the masked comparison on a subset of coefficients is not sensitive information.

We will first detail how plaintext checking oracles, i.e. an oracle that tells if a ciphertext is decrypted correctly so that $m = m'$, can assist in full key recovery. As in [GJN20], we target the specific binary information from a decryption failure oracle in the FO validation check. In contrast to this work which targets FrodoKEM, we also consider schemes that use

[4] The reader might substitute $\boldsymbol{A}_i^{(j)}$ by $\boldsymbol{A}_m^{(j)}[i]$ in Algorithm 8 to resolve this.

ciphertext compression such as Kyber. This compression prohibits gaining exact equations in the secret in our case, which diminishes the information we retrieved from the oracle. To counter this, we show that we can still retrieve approximate equations in the secret key which, combined with the leaky-LWE framework of Dachman-Soled et al. [DDGR20], allows to effectively retrieve the secret key.

Finally, we describe how to obtain efficient plaintext checking oracles for implementations that use the masked FO validation checks of [OSPG18] and [BPO+20]. For the latter, on top of an oracle assisted by side-channel traces, we construct an oracle that takes advantage of collisions in the validation check, and that does not need any side-channel information. As opposed to the attacks in literature, due to the side-channel protections in place, we have to limit the attack to modifying only one coefficient of a valid ciphertext. The reason for this restriction will become clear later.

3.1 Generic key recovery from decryption failure oracles

An inherent feature of many lattice-based encryption schemes is the possibility of decryption failures and the resilience to noisy ciphertexts. Algorithm 3 shows the CPA-secure decryption of Kyber. Momentarily disregarding the compression and NTT operations, the decryption is essentially a two-step approach. First, the decompressed ciphertext components (u, v) are combined with the secret key s to find the intermediate polynomial x':

$$x' = v - s^T u = w + \lceil q/2 \rfloor \cdot m \bmod^{(\pm)} q. \tag{7}$$

In this equation, w is a secret error term that has components both due to compression, as well as the sampled noise elements of MLWE. We refer to [SAB+20] for the full details. After this first step, x' is compressed as in Equation 1 to find m':

$$m' = \text{Compress}_q(x', 1) = \lceil \frac{2}{q} x' \rfloor \bmod^{(+)} 2. \tag{8}$$

In $\text{Compress}_q(x', 1)$, each coefficient of the polynomial x' gets decoded into exactly one message bit. If $-q/4 \leq x'[i] < q/4$, the resulting message bit $m'[i]$ is zero, and when the inverse is true, $m'[i]$ is equal to one.

This means that a decryption failure indicates:

$$(v - \lceil q/2 \rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] < -q/4, \text{ or:} \tag{9}$$

$$(v - \lceil q/2 \rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] \geq q/4. \tag{10}$$

for at least one coefficient i. Parameters are typically chosen such that a decryption failure for valid ciphertexts happens with only negligible probability.

The polynomials u, v and m are part of the input and are typically known to the adversary. Fluhrer [Flu16] showed that knowledge of whether m' equals m, i.e. a decryption failure, is enough to recover the secret key s by successively adapting u, v and observing whether a decryption failure occurs.

Due to practical constraints, we will limit our attack to adapting only one coefficient of a valid ciphertext. A possible attack would proceed as follows. Consider an adversary that submits noisy input ciphertexts $(u, v + e \cdot X^i)$ for a given $i \in [0, n-1]$, where u and v are correctly generated, but with $e \cdot X^i \in R_q$ an additional noise term with only one non-zero coefficient e. In Equation 7, this noise term will appear as an additive term to the secret error term w:

$$x' = (v + e \cdot X^i) - s^T u = (w + e \cdot X^i) + \lceil q/2 \rfloor \cdot m, \tag{11}$$

which leads to the decryption failure conditions:

$$(v - \lceil q/2 \rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] < -q/4 - e, \text{ or:} \tag{12}$$

$$(v - \lceil q/2 \rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] \geq q/4 - e. \tag{13}$$

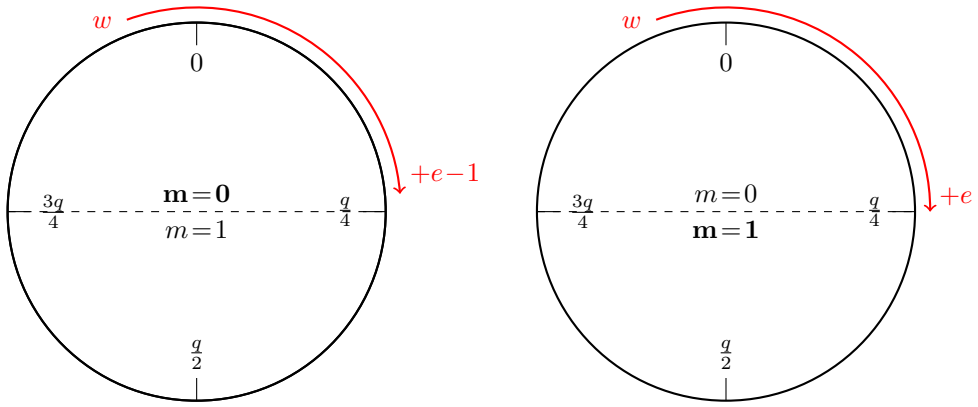


Figure 2: Visualization of a border-failure error e in KYBER.CPA.DEC without compression.

Note that we assume a failure will in this scenario only happen at the adapted coefficient i, since the failure probability of valid ciphertexts is negligible.

An adversary that can submit multiple such ciphertexts, can adaptively tweak e to look for decryption failures. If he finds an error e that does trigger a decryption failure, while $e-1$ does not trigger a decryption failure, he learns the following equation in the secret:

$$(v-\lceil q/2\rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] = \lfloor q/4\rfloor - e. \tag{14}$$

Similarly, an error e that does trigger a decryption failure, while $e+1$ does not, indicates:

$$(v-\lceil q/2\rfloor \cdot m - s^T u \bmod^{(\pm)} q)[i] = -\lceil q/4\rceil - (e+1) \tag{15}$$

We will call a polynomial $e \cdot X^i$ that triggers one of these two conditions a border-failure error. Such a border-failure error is illustrated in Figure 2 for $m[i]=0$, where the error term with coefficient e results in a decryption failure ($m'[i]=1$), but the error term with $e-1$ still results in a correct decryption ($m'[i]=0$).

By performing a binary search as discussed in [GJN20], one can find a border-failure error, and thus an exact equation, in at most $\lceil \log_2(q)\rceil$ iterations (e.g. 12 iterations for Kyber768), by carefully selecting the value of e_1 in each iteration to divide the search space in two. Further, by obtaining kn independent equations, it is possible to fully retrieve the secret error term w and to construct exact equations in the coefficients of s.

In theory, the Fujisaki-Okamoto transformation [FO99] of IND-CCA2-secure lattice-based encryption schemes prevents such attacks by checking if a ciphertext is valid and rejecting adapted ciphertexts. However, an adversary that is able to construct a plaintext checking oracle would be able to circumvent the security of the FO transform and execute this attack.

Compression An additional challenge occurs when applying this technique for key recovery of schemes with compression (e.g. Kyber), which is not covered by [GJN20]. For schemes that do not perform compression on the ciphertext term v it is possible to obtain exact equations in the secret key as explained above. Compressed ciphertexts, on the other hand, are of the form

$$c_1 = \text{Compress}_q(u, d_u) \tag{16}$$

$$c_2 = \text{Compress}_q(v, d_v). \tag{17}$$

This obstructs an adversary to input fine-grained errors e in v, as these small errors would be removed due to the compression.

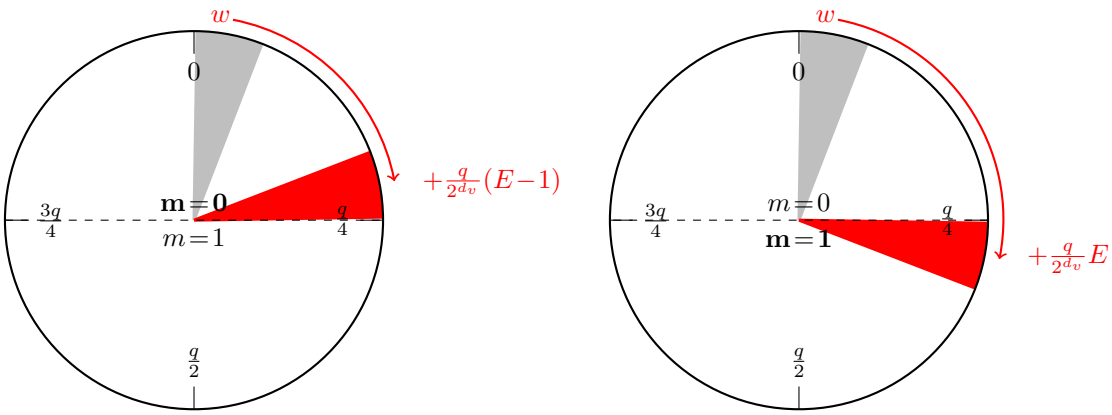


Figure 3: Visualization of triggering decryption error in KYBER.CPA.DEC with error E_2 with compression.

However, it is possible to inject errors E into the term c_2, which will result in coarse-grained errors $e \approx q/2^{d_u} E$ in the decompressed ciphertext after compression.

$$v + e \cdot X^i = \text{Decompress}_q((c_2 + E \cdot X^i), d_v), \tag{18}$$

This means that the space of possible noise values for e is effectively reduced to rounded multiples of $\frac{q}{2^{d_v}}$.

Ciphertext compression thus prevents us from freely choosing the noisy input ciphertext $v + e \cdot X^i$, which in turn prevents us from getting exact equalities. However, similar to the procedure above we can search for values of E, where E does trigger a decryption failure while $E-1$ does not trigger a failure. From such an observation we learn approximate equalities in the secret of the form:

$$\lfloor \frac{q}{2^{d_v}} \cdot (q/4 - E) \rceil \leq (v - \lceil q/2 \rfloor \cdot m - s^T u)[i] < \lfloor \frac{q}{2^{d_v}} \cdot (q/4 - (E-1)) \rceil, \tag{19}$$

which can be written as an approximate equation:

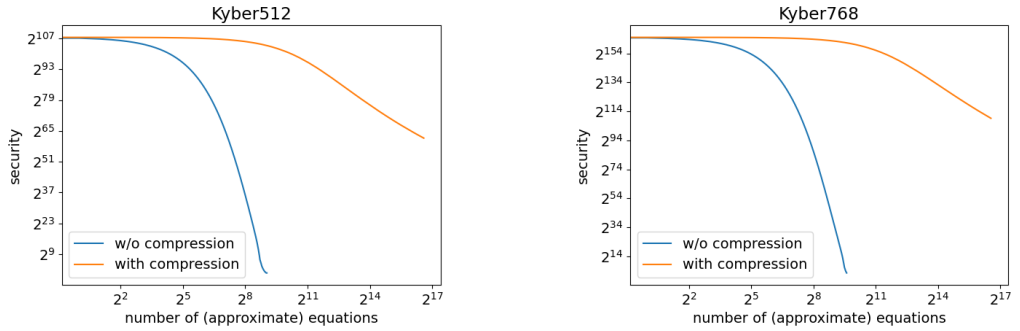
$$(v - s^T u)[i] \approx \lfloor \frac{q}{2^{d_v}} \cdot (q/4 - E + 1/2) \rceil. \tag{20}$$

Figure 3 visualizes the injection of coarse-grained errors when compression is in place. Given a border-failure error E as explained above, w can take any value in the light gray marked interval.

Solving Mod-LWE using (approximate) equations Dachman-Soled et al. [DDGR20] introduced a tool to estimate the security of an LWE sample in the presence of linear hints about the secret. We can use this framework to include our exact or approximate linear equations and estimate the remaining cost of retrieving the secret. In the case of approximate equations, we consider an approximation error variance equal to the variance of a uniform distribution with width $q/2^{d_v}$. Figure 4 gives the security of Kyber512 and Kyber768 after inclusion of approximate or exact hints, which correspond to compressed and non-compressed ciphertexts respectively.

3.2 Side-channel attacks

We will now describe how to construct a plaintext checking oracle that can be used for the attack described before. During the FO transformation, a validation check $c = c'$ is used



(a) Kyber512 **(b)** Kyber768

Figure 4: Security of Kyber512 and Kyber768 in function of the number of (approximate) equations retrieved.

to detect malformed ciphertexts. As mentioned in Section 2.3, the final accept/fail result bit of this check is not sensitive information. This property is used in both [OSPG18] and [BPO+20], where this bit is unmasked in the implementation.

Yet, while the final result bit of the check is not sensitive, information on which part of the message fails certainly is sensitive. Consider, as before, input ciphertexts of the form $c=(c_1,c_2+E\cdot X^i)$ with $E\cdot X^i$ again polynomial with only one nonzero coefficient E. When E is sufficiently small so that $m'=m$, decryption succeeds and $c'=(c'_1,c'_2)=(c_1,c_2)$. On the other hand, when E causes a decryption failure, $m'\neq m$ will essentially randomize the re-encryption through the hashing step in the FO transform (Figure 1). Similar to [GJN20], our decryption failure oracle constitutes visible side-channel leakage between $\text{COMPARE}((c_1,c_2+E\cdot X^i),(c_1,c_2))$ and $\text{COMPARE}((c_1,c_2+E\cdot X^i),(\text{rand}_1,\text{rand}_2))$. We will show that this leakage is present in both [OSPG18] and [BPO+20] because they unmask *partial* checks.

In [OSPG18], it is noted that the different ciphertext components c_1 and c_2 have different sensitivity on the decrypted message m'. The authors argue that m' is only sensitive for an invalid c_1. Consequently, they propose to first check the validity $\text{COMPARE}(c_1,\mathbf{c'_1})$, with COMPARE as in Algorithm 7, and only conditionally execute $\text{COMPARE}(c_2,\mathbf{c'_2})$ in a second step. However, in the plaintext checking oracle outlined above, c_1 *is* a valid ciphertext, since the error is injected into $c_2+E\cdot X^i$. Moreover, the output of $\text{COMPARE}(c_1,\mathbf{c'_1})$ is sensitive, since $m'=m$ results in $check(c_1,\mathbf{c_1})$, whereas in the case $m'\neq m$ this routine will be $\text{COMPARE}(c_1,\mathbf{rand'_1})$. Even though the distinguishable input $\mathbf{c'_1}=\mathbf{c_1}$ or $\mathbf{c'_1}=\mathbf{rand}_1$ is initially masked, the final output of this check is unmasked, and reveals the occurrence of a decryption failure.

The masked comparison of [BPO+20] leaves open a very similar decryption failure oracle to that of [OSPG18]. In Algorithm 8, the comparison proceeds in *sets* of l coefficients, where the pass/fail bit is unmasked for every set. Again, the occurrence of a decryption failure will be readily visible in the side-channel information, since $\text{COMPARE}((c_1,c_2+E\cdot X^i),(c_1,c_2))$ will fail only in the set that contains coefficient i, whereas $\text{COMPARE}((c_1,c_2+E\cdot X^i),(\text{rand}_1,\text{rand}_2))$ will likely fail in all of the sets. Note that for our implementation of Kyber768, since the [BPO+20] requires a prime modulus, COMPARE is not directly instantiated with `MaskedSum`, but must first decompress the ciphertext elements into R_q.

Practical attack To experimentally verify the decryption failure oracles outlined above, we integrated the masked comparisons of [OSPG18] and [BPO+20] into the Kyber768 ARM Cortex-M4 implementation available in PQM4 [KRSS]. The masked comparison of [BPO+20] requires fresh randomness within the algorithm. When this randomness is sampled using rejection sampling, collected traces are misaligned due to the variable timing of this routine. Therefore, to simplify evaluation, we sample all fresh randomness in advance

and pass it to the masked comparison routine as an input parameter.

We use an STM32F407VGT6 chip, mounted on a custom PCB to facilitate power measurements. This custom PCB results in a very stable behavior of the STM32F407 chip. It contains a dedicated shunt resistor to monitor the instantaneous power consumption but is otherwise stripped of unnecessary components that would introduce additional noise into the measurements. The PCB is driven by an external power supply at 3.6 V and clocked by an external clock at 8 MHz, to get maximum stability.

To collect power traces, we use Tektronix DPO 70404C digital oscilloscope and set it to sample power traces at 65 MSamples/s. A PA 303 SMA pre-amplifier performs analog preprocessing of the collected traces before they enter the oscilloscope. A central PC is used to communicate input/output data to the board through a serial USART connection, as well as to collect and analyze power measurements.

We use *Welch's t-test* to detect differences in the power consumption between two classes of measurements. This test computes the so-called t-statistic for every sample in the measurements as:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}}},$$

where $\overline{X}_1$ and $\overline{X}_2$ denote the means of each class, σ_1^2 and σ_2^2 their respective variances, and N_1, N_2 the number of samples.

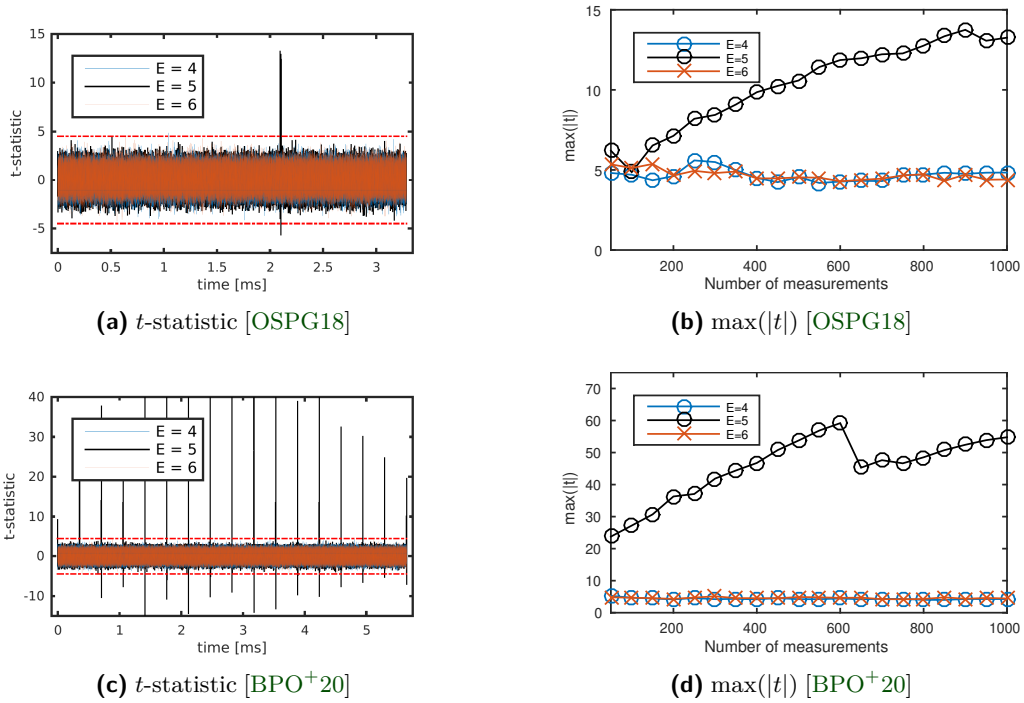
We construct the t-test classes as the input ciphertexts $(c_1, c_2 + E \cdot X^i)$ and $(c_1, c_2 + (E-1) \cdot X^i)$. We specifically target the masked comparison step, where as explained above, we expect t-test leakage when E triggers a decryption failure ($c' = (\mathbf{c}_1, \mathbf{c}_2)$) but $E-1$ does not ($c' = (\mathbf{rand}_1, \mathbf{rand}_2)$). This leakage would allow us to construct approximate equations in the coefficients of the secret key, reducing the R-LWE security.

We generated a pseudorandom Kyber ciphertext from an input seed and conduct the attack described above, with results illustrated in Figure 5. From this figure, it can be deduced that $E = 5$ is the border-failure error. When $E = 4$, both E and $E-1 = 3$ are too small to trigger decryption failures, and they do not leak in the output of the partial comparisons. Conversely, when $E = 6$, both E and $E-1 = 5$ trigger a failure, also preventing partial comparison output leakage. In the middle case where $E = 5$, $E-1$ decrypts correctly but E triggers a decryption failure, and there are large leakage spikes that can be observed. As expected, for [OSPG18] there is a single peak, which we pinpointed to the unmasking of the partial comparison $\text{COMPARE}(c_1, \mathbf{c}'_1)$. On the other hand, for [BPO+20] there are 16 peaks corresponding to the unmasking of the $l = 16$ partial sets. Our attacks require a very limited amount of traces to detect the border-failure error. When looking at the maximum absolute t-statistic value over the measurements on the right side of Figure 5, the border-case failure is detectable after a few hundred traces for [OSPG18] and already after the first block of 50 traces for [BPO+20].

To further illustrate why $E = 5$ is the border-failure error, we inspected internal variables. For our pseudorandom Kyber ciphertext it holds that:

	$E=0$	$E=4$	$E=5$
$c_2[0]$	5	9	10
$v[0] + e = \lfloor \frac{q}{2^{d_v}} \cdot c_2[0] + E \rceil$	1040	1873	2081
$(v - s^T u)[0] + e$	-55	778	986

Since $986 \geq q/4 = 832.25$, the ciphertext decrypts to $m'[0] = 1$, such that indeed $E = 5$ is the border-case failure error term. From the side-channel information, an adversary that does not know $(v - s^T u)[0] = -55$ can now compute it approximately, and finds that: $(v - s^T u)[0] \approx -105$.



(a) t-statistic [OSPG18] **(b)** max($|t|$) [OSPG18]

(c) t-statistic [BPO+20] **(d)** max($|t|$) [BPO+20]

Figure 5: Testing for border-failure errors using a t-test with the input ciphertext classes $(c_1, c_2 + E \cdot X^i)$ and $(c_1, c_2 + (E-1) \cdot X^i)$ for the masked comparisons of [OSPG18] and [BPO+20].

3.3 Collision attack

In this section, we show that it is possible to extract information on the Kyber secret key using a reaction attack without side-channel information when the method from [BPO+20] is used for comparison in a masked implementation, thus breaking the CCA security of the scheme. Firstly we recall the four scenarios an attacker may experience when querying a decryption oracle. Assume c_a is a valid ciphertext, c is the re-encrypted ciphertext and $\tilde{c}_a$ is the ciphertext that is sent to the oracle by the attacker.

1. $\tilde{c}_a = c_a$: A valid ciphertext has been provided such that $m = m'$ and thus the re-encryption uses the same random coins r as the original encryption. The result of the final comparison $c = \tilde{c}_a$ is *true*.

2. $\tilde{c}_a = c_a + e$, where e is a polynomial with a limited number of small coefficients: A valid ciphertext is slightly modified by a small error but in a way that the decryption still results in $m = m'$. The result of the final comparison $c = \tilde{c}_a$ is *false* but the difference between the two ciphertexts is small as $e = \tilde{c}_a - c$. In the rest of this subsection we will consider e a polynomial with only one non-zero coefficient, which means that the difference between $\tilde{c}_a$ and c is limited to only one coefficient.

3. $\tilde{c}_a = c_a + e$, where e is a polynomial with one or more large coefficients: A valid ciphertext is modified in such a way, that $m \neq m'$ after decryption (e.g., in one bit). Thus the re-encryption operates on completely different random coins r than the original encryption. Internal variables of the re-encryption differ from the original encryption. The result of the final comparison $c = \tilde{c}_a$ is *false* and the difference of $\tilde{c}_a$ and c is very large as c can be considered to be randomly generated due to the different coins.

4. $\tilde{c}_a$ is uniform: A completely unrelated and invalid ciphertext is sent to the decryption oracle. Same result as when e is large.

The masked comparison of [BPO+20] has a certain probability of false positives, where invalid ciphertexts are still accepted. These false probabilities are denoted collisions and the authors analyzed that they occur with probability $1/q^x$ with x being the number of comparison sets used. In [BPO+20] the authors recommend a value of $x = 16$ for Kyber. The probability of a collision in a single set $P_{single-coll}$ was shown to be $1/q$.

However, their analysis included scenarios 1, 3, and 4, but did not take into account the possibility of scenario 2, where only one coefficient and thus only one set fails. In this scenario, the false positive probability equals the single set collision probability $1/q$. This relatively high collision probability is what we exploit in our attack.

In practice, our attack queries the decryption failure oracle many times with a slightly modified ciphertext $(c_1, c_2 + E \cdot X^i)$ as in Subsection 3.1. As before, $E \cdot X^i$ is a polynomial that is non-zero except for its i^{th} coefficient E. If no decryption failure occurs we are in scenario 2 and the scheme will return a correct decryption of the original ciphertext with probability $P_{single-coll}$. If it did cause a faulty decryption, we are in scenario 3 and the ciphertext $(c_1, c_2 + E \cdot X^i)$ will not accept the message[5]. This means that when the ciphertext $(c_1, c_2 + E)$ is accepted we know that $m = m'$, and this acceptance happens with probability $1/q$.

However, when $m \neq m'$ we do not get any definite proof as the ciphertext will always be rejected. In this case, we can construct an error term $\overline{E}$ where:

$$\overline{E} = E - 2^{d_v - 1} \bmod^{(+)} 2^{d_v} \tag{21}$$

which results in an error:

$$\overline{e} \approx e - q/2 \bmod^{(+)} q. \tag{22}$$

The effect of the term $q/2$, is to flip the message, and x' becomes:

$$x'[i] = (v + \overline{e})[i] - s^T u[i] = w[i] + e + \lceil q/2 \rfloor \cdot (m[i] \oplus 1), \tag{23}$$

This way, $(c_1, c_2 + \overline{E} \cdot X^i)$ behaves in the opposite fashion: $m = m'$ will always lead to a rejected ciphertext, while $m \neq m'$ will succeed with probability $1/q$.

On average, the decryption oracle has to be queried $2q$ times, q times with error E and q times with error $\overline{E}$, to obtain one correctly decrypted message from the original ciphertext which indicates whether $m = m'$ or $m \neq m'$. As such we have successfully constructed a decryption failure oracle that can be used in an attack as outlined in Subsection 3.1. Analogously to before, we aim to find the border-failure error E so that $E - 1$ does not trigger a decryption failure but $E + 1$ does.

We verified the attack in practice using the PQClean implementation of Kyber768 on Intel Core i5 - 8350U CPU with 1.7 GHz. Without the compression of Kyber, recovery of a border-failure error took around 3 minutes on average. That includes around $2^{16.2}$ calls to the decryption oracle, which is a combination of the binary search cost $\log_2(q)$ and the cost of finding one border-failure $2q$. As can be seen in Figure 4, obtaining 2^{11} equations leads to a full recovery of the secret key in this case. In conclusion, for Kyber768 without compression, it would be possible to retrieve the full secret key in $2^{27.2}$ calls to decapsulation. This attack would take roughly 100 hours on our device.

For Kyber512 or Kyber768 with compression, we are expecting to find an approximate equation with $2q\log_2(2^{d_v}) \approx 2^{13.7}$ queries, where 2^{d_v} is the number of values E can take and $2q$ again the cost to find a collision. The cost of the remaining security can be calculated by [DDGR20] and is graphically presented in Figure 4. All in all, the number of queries and remaining computational effort for Kyber768 is still high. However, for a fast decryption oracle, an attack onto Kyber512 with $2^{13.7} \cdot 2^{17} = 2^{30.7}$ queries to obtain 2^{17} equations that lead to a remaining computational complexity of 2^{65}, which seems within practical reach.

[5] To be more accurate, the message is accepted with probability $1/q^x$ (e.g. 2^{-186} for Kyber768), which we can assume will not happen in practice.

4 Correcting [BPO+20]

The first order masked comparison of [OSPG18] can be fixed easily by combining both c_1 and c_2 in a single hash for comparison as done in [BDK+20]. The higher-order masked comparison of [BPO+20] is not straightforward to fix. One option is to fall back to a generic comparison where all coefficients of the ciphertext are first converted into a Boolean sharing after which the comparison is performed using an appropriate masked Boolean circuit. This would require kn A2B conversions for c_1 and n for c_2, which is a total of $l_A = (k+1)n$ A2B conversions, where n is the number of coefficients of the polynomials in the ring R_q and where k is the number of polynomials in the secret vectors. Additionally one needs an appropriate Boolean circuit to perform the comparison on the masked Boolean shares.

It is however possible to use the idea of [BPO+20] to reduce the number of A2B conversions needed. The idea is to combine all coefficients in one masked sum, and to perform a masked comparison on only this sum instead of all coefficients separately. This would lead to a masked comparison that only needs one A2B conversion, but with a probability of a false positive (i.e. the comparison returns true while the inputs are not equal) of $1/q$. To reduce the false positive probability one can then repeat this procedure l_B times, leading to a false positive probability of $1/q^{l_B}$. In such scenario, it is important that the final l_B comparisons are performed in a secure way so that only the final output of all comparisons combined is outputted and no intermediate comparison results are leaked.

To provide security and avoid the problems in [BPO+20], we introduce two significant changes to the original proposal of [BPO+20]: First, all coefficients are combined in every masked sum $\mathbf{B}$, to avoid the attacks as described in the previous section. Secondly, we generate the masked sum $\mathbf{B}$ in S shares instead of just 1 share in the original proposal, which is a requirement to obtain $(S-1)^{\text{th}}$ order security.

Algorithm 9 gives an algorithm that compresses the $l_A = (k+1)n$ coefficients that need to be compared in a side-channel secure way, to l_B coefficients that need to be compared in a side-channel secure way. We will first prove that the l_B remaining coefficients give the same result after comparison as the initial l_A coefficients, except for a false positive probability of $1/q^{l_B}$. We will then prove the security of our algorithm. An experimental validation of our algorithm can be found in Appendix B.

Algorithm 9: `ReduceComparisons`

Input : A set $\boldsymbol{A}$ consisting of shared coefficients $\boldsymbol{A}_1,\ldots,\boldsymbol{A}_{l_A} \in \mathbb{Z}_q^{(S)}$ such that $\sum_{j=1}^{S} \boldsymbol{A}_{i_A}^{(j)} = A_{i_A} \bmod q$; A set $\tilde{A}$ consisting of coefficients $\tilde{A}_1,\ldots,\tilde{A}_{l_A} \in \mathbb{Z}_q$

Output : An m-th subset $\boldsymbol{B}$ consisting of shared coefficients $\boldsymbol{B}_1,\ldots,\boldsymbol{B}_{l_B} \in \mathbb{Z}_q^{(S)}$; A set $\tilde{B}$ consisting of coefficients $\tilde{B}_1,\ldots,\tilde{B}_{l_B} \in \mathbb{Z}_q$

for $i_B = 1$ **to** l_B **do**
  $(\mathbf{B}_{i_B}^{(j)})_{1 \le j \le n} \leftarrow 0$
  $\tilde{B}_{i_B} \leftarrow 0$
  for $i_A = 1$ **to** l_A **do**
    $R \xleftarrow{\$} \mathbb{Z}_q$
    for $j = 1$ **to** S **do**
      $\mathbf{B}_{i_B}^{(j)} \leftarrow \mathbf{B}_{i_B}^{(j)} + \mathbf{A}_{i_A}^{(j)} \cdot R \bmod q$
    end
    $\tilde{B}_{i_B} \leftarrow \tilde{B}_{i_B} + \tilde{A}_{i_A} \cdot R \bmod q$
  end
end
return $\mathbf{B}_1,\ldots,\mathbf{B}_{l_B},\tilde{B}_1,\ldots,\tilde{B}_{l_B}$

Theorem 1. *Let n, r be positive integers, let q be a prime. Then after execution of* `ReduceComparisons`*, the comparison of $\mathbf{A}_1,...,\mathbf{A}_{l_A}$ with their respective values $\tilde{A}_1,...,\tilde{A}_{l_A}$ gives the same result as the comparison of $\mathbf{B}_1,...,\mathbf{B}_{l_B}$ with their respective values $\tilde{B}_1,...,\tilde{B}_{l_B}$, with the exception of a false positive probability of $1/q^{l_B}$. More specifically:*

- *If $1 < i_A \leq l_A : \sum_{j=1}^{S} \mathbf{A}_{i_A}^{(j)} = \tilde{A}_{i_A}$, then $1 < i_B \leq l_B : \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} = \tilde{B}_{i_B}$*
- *If $\exists_{i_A} \sum_{j=1}^{S} \mathbf{A}_{i_A}^{(j)} \neq \tilde{A}_{i_A}$, then $\exists_{i_B} \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} \neq \tilde{B}_{i_B}$ with probability $1 - 1/q^{l_B}$*

Proof. We first calculate the values of $\mathbf{B}_{i_B}$ and $\tilde{B}_{i_B}$. Secondly, we will reason about the case where the all coefficients of $\mathbf{A}_{i_A}$ and $\tilde{A}_{i_A}$ match, and finally we will look at the case where at least one of these pairs does not match.

We will denote with $R^{[i_B, i_A]}$ the random value R generated in the i_B^{th} round of the outer loop and the i_A^{th} of the second loop. Then, by simple substitution, one can show that:

$$\mathbf{B}_{i_B}^{(j)} = \sum_{i_A=1}^{l_A} \mathbf{A}_{i_A}^{(j)} \cdot R^{[i_B, i_A]}$$

$$\tilde{B}_{i_B} = \sum_{i_A=1}^{l_A} \tilde{A}_{i_A} \cdot R^{[i_B, i_A]}$$

which means that:

$$\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} = \sum_{i_A=1}^{l_A} \left(\tilde{A}_{i_A} - \sum_{j=1}^{S} \mathbf{A}_{i_A}^{(j)} \right) R^{[i_B, i_A]}$$

Equal inputs In the case where the two inputs match (i.e. $\forall_{i_A} \sum_{j=1}^{S} A_{i_A}^{(j)} = \tilde{A}_{i_A}$), we have:

$$\tilde{A}_{i_A} - \sum_{j=1}^{S} \mathbf{A}_{i_A}^{(j)} = 0,$$

and thus:

$$\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} = 0,$$

or equally:

$$\tilde{B}_{i_B} = \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)},$$

which confirms our first statement.

Different inputs This leaves the case where at least one of the input coefficients does not match (i.e. $\exists_{i_A} \sum_{j=1}^{S} A_{i_A}^{(j)} \neq \tilde{A}_{i_A}$). Without loss of generality we assume that this is the case for $i_A = 1$, (i.e. $\sum_{j=1}^{S} A_1^j \neq \tilde{A}_1$), so that we can rewrite the input to the masked comparison as:

$$\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} = \left(\tilde{A}_1 - \sum_{j=1}^{S} \mathbf{A}_1^j \right) R^{[i_B, 1]} + \sum_{i=2}^{l_A} \left(\tilde{A}_{i_A} - \sum_{j=1}^{S} \mathbf{A}_{i_A}^{(j)} \right) R^{[i_B, i_A]}.$$

The first term $\left(\tilde{A}_1 - \sum_{j=1}^{S} \mathbf{A}_1^j \right) R^{[i_B, 1]}$ is a multiplication of a nonzero term with a uniformly random term. As multiplication with a nonzero field element is a bijection, the distribution

of this first term is also uniform over the field. Addition of the other terms does not change the uniformity of the distribution and therefore $\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ is uniformly distributed.

A uniform distribution of $\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ means that $\tilde{B}_{i_B} = \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ with probability $1/q$ (i.e. when $\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)} = 0$), and that $\tilde{B}_{i_B} \neq \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ with probability $(q-1)/q$. Moreover, note that the values of $\tilde{B}_{i_B} - \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ are independent for different i_B, as the randomness $R^{[i_B,1]}$ varies in each iteration.

Due to this independence, the probability that for all $i_B \in [1,l_B]$ it holds that $\tilde{B}_{i_B} = \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ is $1/q^{l_B}$. Taking into account that probabilities sum to one, the probability that at least for one $i_B \in [1,l_B]$ it holds that $\tilde{B}_{i_B} \neq \sum_{j=1}^{S} \mathbf{B}_{i_B}^{(j)}$ is $1-1/q^{l_B} = 1-1/q^{l_B}$. □

Theorem 2. *Algorithm 9 is t-NI at any order $t<n$.*

Proof. The essence of the proof is that in Algorithm 9, different shares of the same input variable are never combined. Every intermediate and output value depends on maximal one share of each input variable. The intermediate values are $R^{[i_B,i_A]}$, $\tilde{B}_{i_B}^{[i_A]}$ and $\mathbf{B}_{i_B}^{(j),[i_A]}$. First, the randomness $R^{[i_B,i_A]}$ can be simulated uniformly at random from $\mathbb{Z}_q$. Secondly, any intermediate value $\tilde{B}_{i_B}^{[i_A]}$ is constructed without involvement of the sensitive input values and therefore are perfectly simulatable without knowing any input share, by following the exact procedure in Algorithm 9. Finally any intermediate value of $\mathbf{B}_{i_B}^{(j),[i_A]}$ only depends on the j^{th} shares of $\mathbf{A}_{i_A}$ and thus can be simulated without knowledge of any $\mathbf{A}_{i_A}^{t}$ where $t \neq j$. □

After the application of *Algorithm 9*, the number of coefficients that need to be compared is reduced from $l_A = (k+1)n$ to l_B, where l_B should be chosen so that the false positive probability $1/q^{l_B}$ is small enough to avoid that an adversary can practically find such false positives. For Kyber768, when a false positive probability of maximum 2^{-128} is required, this implies a reduction from $l_A = 1024$ coefficients to $l_B = 11$ coefficients that need to be compared.

However, the constraints of [BPO+20] still apply to our proposed correction, mainly the fact that q needs to be prime. Schemes with power-of-two moduli or schemes that perform compression of ciphertexts to power-of-two moduli, as is frequently the case in lattice-based encryption schemes, need an additional masked operation to convert the power-of-two shares to prime moduli. This might complicate the usage of our method, or might in extreme cases make it more expensive than the generic masked comparison. The application of *Algorithm 9* is therefore scheme-specific and as it is not the main target of this paper, we refrain from an extensive treatment here and leave this for future work.

5 Framework for side-channel testing

The masking schemes that were attacked in Section 3 were tested by the original authors for side-channel information leakage using Test Vector Leakage Assessment (TVLA) [GJJR11]. A natural question is then why this side-channel leakage did not show up in the test results and how to develop a test that would capture these leakages, which can be used to replace the traditional test.

A standard scenario for testing is a fixed vs random (FvR) test, in which the fixed category consists of valid input ciphertexts and in which the random test set comprises of uniform random inputs $u,v \leftarrow U(q)^{k\times 1} \times U(q)$. Remember that the result of the comparison is allowed to be leaked, but no other information besides this one bit should be leaked. As the comparison output is different between both testing sets, there is potential for non-malevolent output leakage showing up in the t-test. While this output leakage can be ignored, it is hard to assess if no real leakage is ignored in the same breath. For example, in [BDK+20, Figure 8], this output leakage is shown, and the authors argue that it is non-sensitive.

Fixed + Noise	Random
$(c_1,c_2) \leftarrow$ KYBER.CPA.ENC(pk,m,r)	
while True:	while True:
$(c'_1,c'_2) \leftarrow (c_1,c_2)$	$(c'_1,c'_2) \xleftarrow{\$} U(d_u)^{k\times 1} \times U(d_v)$
$pos \xleftarrow{\$} [0,(k+1)\cdot n)$	return (c'_1,c'_2)
$(c'_1,c'_2)[pos] \leftarrow (c_1,c_2)[pos]+1$	
return (c'_1,c'_2)	

Figure 6: Generation of inputs ciphertexts following the classes used in the FNvR test.

In this section, we develop a new test method that has the same leakage detection capacity as the standard FvR test, but that eliminates the output leakage. We will formalize our new method both for testing the masked comparison and for testing the whole decapsulation. Our framework scales to arbitrary orders similar to the traditional t-test.

5.1 'Fixed + noise' vs random

Our method uses a 'Fixed + Noise' vs Random (FNvR) test. The key idea is to introduce small random noise to the fixed valid ciphertext so that the comparison output is false in both test scenarios. By keeping the noise small and varying its location at random, the test is still able to find the same vulnerabilities as the traditional FvR method after combining all traces.

Our FNvR test can be understood as a specific instance of a *Semi-Fixed vs Random* (SFvR) t-test. SFvR tests are sometimes used to eliminate false positives related to input- or output leakage in a FvR test. In such a test, rather than fixing the input to a constant value, part of the sensitive intermediate variables is fixed. Consequently, the input ciphertext is not a single fixed representative, but uniformly drawn from the set of ciphertexts that results in the desired intermediate variable values. In this setting, any apparent leakage results from the sensitive variable, rather than fixed inputs or outputs.

Our choice to submit noisy input ciphertext fixes the value $(c \stackrel{?}{=} c') = 0$, thus eliminating the output leakage resulting from the fixed input being a valid ciphertext. To maintain similar leakage detection to a standard FvR test, we keep the noise small, which additionally fixes the internal variable $m = m'$ with high probability. Specifically, the 'fixed + noise' set is generated as a valid ciphertext to which the smallest possible noise value is added to one of the coefficients at random, as shown in Figure 6. A C code listing that implements our FNvR test is provided in the Appendix A, which also takes into account that ciphertexts are encoded into bitstrings.

We note that the proposed FNvR does not suffer from the common pitfalls of SFvR test [UvWBS20]. The complexity of SFvR grows with the size of the set of semi-fixed ciphertext, as all ciphertext in the set must be exhaustively tested. Moreover, the choice of intermediate variable may introduce some bias, making the test leakage model dependent. However, FNvR introduces a small noise in one of the coefficients chosen at random, limiting the complexity increase to the total number of coefficients. Moreover like FvR, FNvR is also performed with valid (+ small noise) and random ciphertext, thus avoiding introducing bias towards any leakage model.

FNvR leakage detection capabilities Our FNvR method is a validation test to detect vulnerabilities in a masked implementation, which eliminates the output leakage present in the traditional FvR test. Here, we make an informal argument why the FNvR test has the same leakage detection capabilities as the FvR test. Consequently, our FNvR test can be used as a direct replacement for the more traditional FvR method. Notwithstanding that many vulnerabilities will be captured by our new test, it is possible that vulnerabilities are not captured by the FvR test and therefore will also not be captured by our FNvR method.

When applying the FNvR test on the decapsulation step, the output leakage is suppressed by introducing a small noise to one of the coefficients of the input ciphertext leading to a comparison that always fails. To reason about the leakage detection capability of the FNvR test compared to the traditional test, one can split the decapsulation into its three phases: first decryption, then re-encryption, and finally the comparison.

1. **Decryption** For decryption, our analysis makes use of the fact that this is a coefficient-wise operation with respect to the polynomial v, or equivalently, the ciphertext part c_2. When an error is injected into $c_2[i]$, all computations on coefficients $j \neq i$ are identical to those in a 'fixed' class of measurements. Since we inject errors randomly into (c_1,c_2), the probability that the error is injected into $c_2[i \neq j]$ is equal to $(n-1)/((k+1)n)$. Each individual coefficient j therefore tends to its 'fixed' value, with a probability of $(n-1)/((k+1)n)$, and accordingly, this 'fixed' value will leak under the same assumptions as a standard FvR test, albeit possibly with more collected traces.

2. **Re-encryption** The re-encryption step only depends on the decrypted message. As such it is identical for ciphertexts from the 'fixed + noise' and the 'fixed' classes, given that no decryption failure occurs due to the extra noise element. To minimize this decryption failure probability, one should choose noise coefficients as small as possible[6], i.e. ± 1. In practical schemes, this should lead to a very small decryption failure probability. For example, in the case of Kyber768 this probability under 'fixed + noise' inputs is 2^{-87}. When a decryption failure would occur, the trace will be identical to a trace in the random class of measurements. Given the very small probability of inserting random traces into the 'fixed+noise' class, sufficient traces without decryption failures will be present in the 'fixed+noise' class.

3. **Comparison** In the comparison, the output leakage is suppressed due to the pass/fail bit resulting in a fail for both classes of measurements. On the other hand, we have to show that FNvR does capture leakage that would be captured by the FvR test. We can follow similar reasoning as for the encryption, namely that the 'fixed + noise' class changes only 1 coefficient of the input ciphertext at random, while all other coefficients remain the same as in the fixed category. This means that both methods will essentially find the same leakage, except when this leakage is at the changed coefficient of the input ciphertext. By varying the location of the coefficient that is changed, one can make sure that leakage from all coefficients is tested.

One can argue that our FNvR test is more realistic than the FvR test, as an adversary is allowed to distinguish between a valid and random ciphertext, but should not be allowed to obtain any other information. The 'fixed + noise' is the closest category to the fixed category that gives invalid ciphertexts.

5.2 Validation of the methodology

We validated our FNvR framework on the masked comparisons of [OSPG18, BDK+20, BPO+20]. We treat `ReduceComparisons` (Algorithm 9) separately in the Appendix, as it is not a full comparison and doesn't compute the pass/fail bit. At the same time, we give a cautionary note regarding input leakage.

Figure 7 shows our results. On the left, we conducted a regular FvR t-test, where the fixed input is a valid ciphertext. All of the masked comparisons show output leakage, due to the output pass/fail bit differing between the fixed and random class of measurements.

On the right, we conduct our novel FNvR t-test. The output pass/fail bit is no longer leaked since it is identical between the two classes of measurements. As expected, we

[6] Without loss of generality, we exclusively inject +1.

can see that the output leakage indeed vanishes from [BDK+20], but remains present in [OSPG18, BPO+20]. Similar to our attack on these two implementations, a noise term added in just one coefficient will cause one partial comparison to fail in the 'fixed+noise' class of measurements, but the remaining partial comparisons will succeed. This stands in contrast with the random class of measurements, where all partial checks will fail with high probability. In [OSPG18], we can now observe a second leakage peak, since an error added to c_1 will still cause the comparison of c_2 to succeed[7]. The noise term is varied over the different coefficients, such that all partial comparisons will eventually show leakage in our framework.

6 Conclusion and Future Work

In this work, we attacked the masked implementations of polynomial comparison from [OSPG18] and [BPO+20] using a side-channel attack. Additionally we demonstrated a collision attack on [BPO+20] that does not require side-channel information. Both attacks were verified in practical experiments. When instantiated with either of these masked polynomial comparison methods, the security of Kyber512 can be reduced drastically using a sufficient number of side-channel measurements (or decapsulation queries for [BPO+20]).

In addition, we showed that the polynomial comparison variant described in [BDK+20] can be used to fix the insecure comparison of [OSPG18] and proposed a modification to the [BPO+20] method that can reduce the number of coefficient comparisons to be performed. Our FNvR framework for side-channel testing suppresses output leakage and thus makes it easier to distinguish between real leakage and output leakage. Moreover, our FNvR test still detects all vulnerabilities a normal FvR t-test would detect.

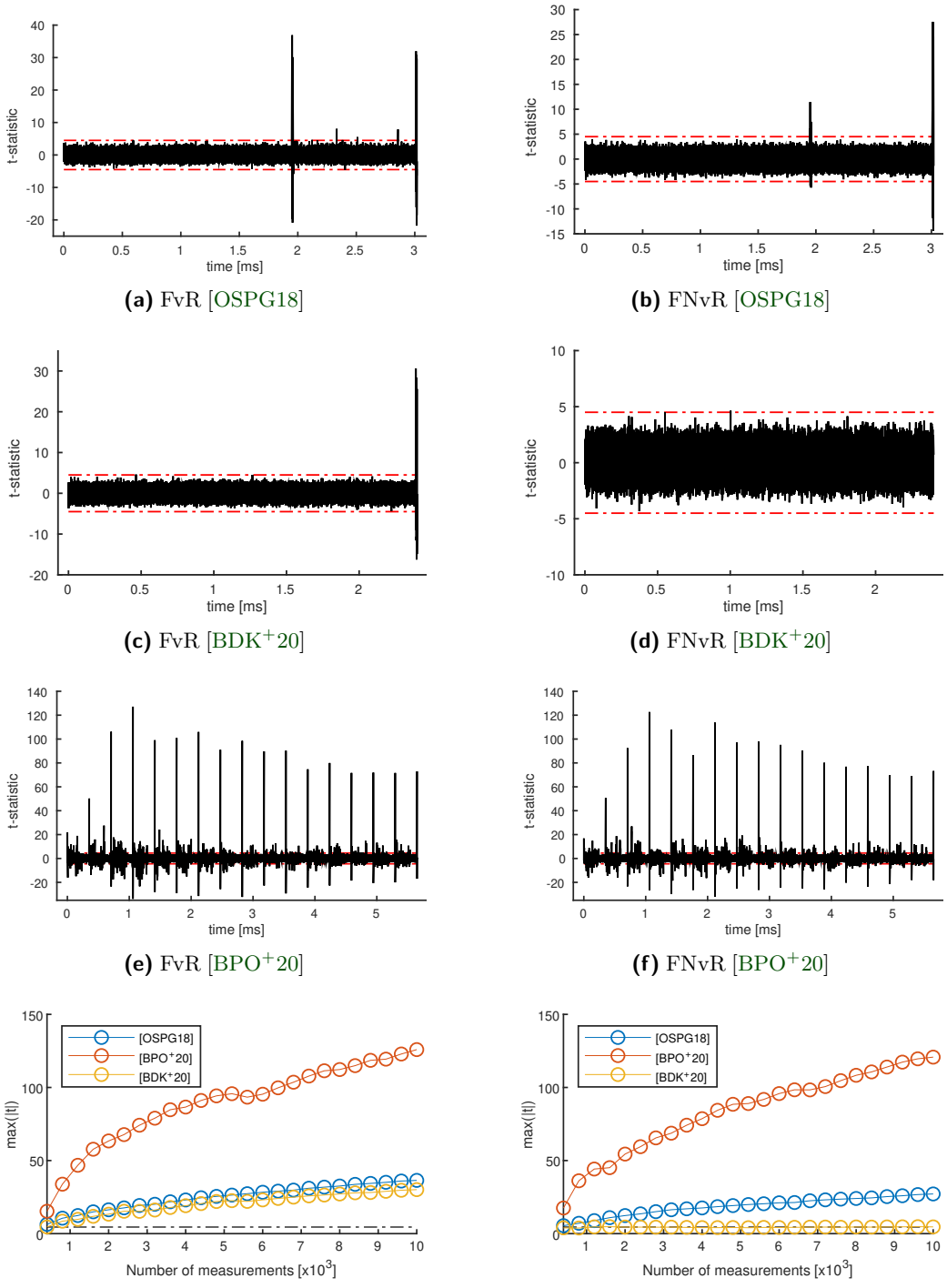
All in all, our work shows that a careful validation and implementation of all steps of the FO transform is required when implementing secured lattice-based cryptography. Small leakages, implementation mistakes, or misunderstandings related to the attack model in the masked comparison might lead to powerful plaintext checking oracles. As a consequence, it is important to investigate new countermeasures, prove their security in the correct model, and develop approaches for theoretical and practical validation. With our FNvR framework, we provide the first step in this direction.

As a recent attack on a first-order masked Saber implementation [NDGJ21] has shown, first-order masking is not enough to provide sufficient protection against side-channel attacks. Therefore, the efficient design and implementation of side-channel protection, including shuffling techniques and higher-order masking, remains an interesting open topic. This concerns the masked comparison but also other building blocks used in the FO transform. Future work in this direction could focus on higher-order masked polynomial comparisons, their efficient implementation, or on algorithms that can support prime and power of two moduli. In particular, it would be interesting to implement our correction of [BPO+20] to investigate the final performance to compare the results with other approaches. Additionally, it would be interesting to investigate improvements or generalizations of the framework provided in [DDGR20] to reduce the number of required approximate linear equations. A good understanding of the cost of retrieving the full secret could help to bound the amount of side-channel leakage an attacker would be allowed to obtain in a realistic scenario.

Acknowledgments

We thank the authors of [OSPG18] and [BPO+20] for providing us with the source code to verify our attacks. Part of this work was done during the Lorentz Center Workshop

[7] Noise terms added to the u component of the ciphertext, $u+e$ will cause errors $s \cdot e$. These will likely not trigger a decryption failure because s is sampled from a distribution of small elements.



(a) FvR [OSPG18] **(b)** FNvR [OSPG18]

(c) FvR [BDK$^+$20] **(d)** FNvR [BDK$^+$20]

(e) FvR [BPO$^+$20] **(f)** FNvR [BPO$^+$20]

(g) FvR max($|t|$) **(h)** FNvR max($|t|$)

Figure 7: t-statistic for the different masked comparison algorithms after 10.000 collected traces, for a FvR test (left) and a FNvR test (right).

"Post-Quantum Cryptography for Embedded Systems", Oct. 2020 and we are thankful to the Lorentz Center, its staff, the organizers and participants of the workshop.

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), SRC grant 2909.001, and the German Federal Ministry of Education and Research (BMBF) under the project "Aquorypt" (16KIS1017). Michiel Van Beirendonck is funded by an FWO PhD fellowship strategic basic research. 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

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the second round of the NIST PQC standardization process. *NIST, Tech. Rep., July*, 2020.

[ACLZ20] Dorian Amiet, Andreas Curiger, Lukas Leuenberger, and Paul Zbinden. Defeating NewHope with a single trace. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 189–205. Springer, Heidelberg, 2020.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS – Kyber: A CCA-secure module-lattice-based KEM. In *EuroS&P*, pages 353–367. IEEE, 2018.

[BDK+20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of SABER. Cryptology ePrint Archive, Report 2020/733, 2020. https://eprint.iacr.org/2020/733.

[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.

[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.

[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*, 2019.

[Flu16] Scott Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085, 2016. http://eprint.iacr.org/2016/085.

[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.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. NIST Non-Invasive Attack Testing Workshop - NIAT, 2011.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Heidelberg, August 2020.

[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.

[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.

[NAB+20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM. *IACR Cryptol. ePrint Arch.*, 2021:79, 2021. https://eprint.iacr.org/2021/079.

[NIS16] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf, 2016.

[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.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, Heidelberg, September 2017.

[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.

[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.

[UvWBS20] Balazs Udvarhelyi, Antoine van Wassenhove, Olivier Bronchain, and François-Xavier Standaert. On the security of off-the-shelf microcontrollers: Hardware is not enough. In *International Conference on Smart Card Research and Advanced Applications*, pages 103–118. Springer, 2020.

A FNvR t-test

```
int main(void)
{
    const int L_A = (KYBER_K+1)*KYBER_N;

    uint8_t sk[KYBER_SECRETKEYBYTES];
    uint8_t pk[KYBER_PUBLICKEYBYTES];
    uint8_t c[KYBER_CIPHERTEXTBYTES]; // valid
    uint8_t K[KYBER_SSBYTES];

    uint8_t cR[KYBER_CIPHERTEXTBYTES]; // random
    uint8_t cFN[KYBER_CIPHERTEXTBYTES]; // 'fixed + noise'
    uint8_t cFNvR[KYBER_CIPHERTEXTBYTES];  // 'fixed + noise' vs random

    int16_t c1c2[L_A];
    uint8_t seed[KYBER_SYMBYTES];
    uint8_t coin, uint16_t pos;

    // Receive seed from central PC.
    // The central PC can recompute and verify all the operations.
    usart_recv_bytes(USART2, seed, KYBER_SYMBYTES);
    init_prng(seed);

    // Generate keypair
    crypto_kem_keypair(pk, sk);

    // Generate valid ciphertext
    crypto_kem_enc(c, K, pk);

    while(1)
    {
        // Sample cR, pos, coin
        randombytes(cR, KYBER_CIPHERTEXTBYTES);
        randombytes(&pos, 2);
        randombytes(&coin, 1);

        // Decode valid ciphertext
        polyvec_decode(c1c2, c);

        // Add noise & encode into cFN
        c1c2[pos % L_A] += 1;
        polyvec_encode(cFN, c1c2);

        // Coin decides the measurement class : cFN <-> cR
        memcpy(cFNvR, (coin % 2) ? (cFN : cR), KYBER_CIPHERTEXTBYTES);

        // Decapsulate cFNvR
        crypto_kem_dec(K, cFNvR, sk);

        // Send K to central PC for verification
        usart_send_bytes(USART2, K, KYBER_SSBYTES);
    }
}
```

B Side-channel validation of ReduceComparisons

We validated our proposed reduction method `ReduceComparisons` (Algorithm 9) using the TVLA methodology. Note that we only implement the reduction method and not the full comparison and as such don't suffer from output leakage. This means that we can apply the standard FvR methodology.

However, the measurements will show non-malevolent input leakage similar to the output

leakage in FvR methods. This is because in line 9, there is a computation on the non-sensitive variable $\tilde{A}$, which, in our case, is the submitted input ciphertext. In our implementation, we moved this computation to the end of the algorithm in a separate loop, and use a yellow trigger to mark it in the measurements. As can be seen in Figure 8, the t-statistics takes high values during this interval, but otherwise shows no leakage during the processing of the $S=2$ shares. These measurement confirm our theoretic analysis of the security of Algorithm 9. We leave the development of techniques to reduce the input leakage for future work.

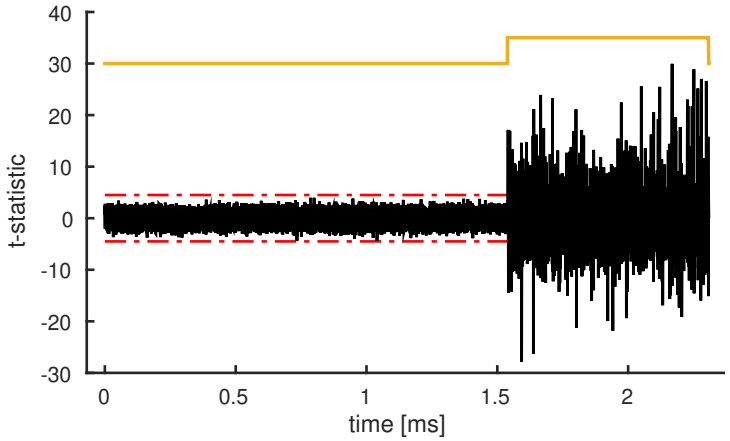


Figure 8: t-statistic for `ReduceComparisons` after 10.000 collected traces, for a FvR test.