The Insecurity of Masked Comparisons: SCAs on ML-KEM's FO-Transform

Julius Hermelink[1], Kai-Chun Ning[1] and Emanuele Strieder[23]

[1] MPI-SP, Bochum, Germany
firstname.lastname@mpi-sp.org
[2] Fraunhofer AISEC, Garching, Germany
emanuele.strieder@aisec.fraunhofer.de
[3] Technical University of Munich, Germany

Abstract. NIST has released the draft standard for ML-KEM, and ML-KEM is actively used in several widely-distributed applications. Thus, the wide-spread use of ML-KEM in the embedded worlds has to be expected in the near future. This makes security against side-channel attacks a pressing matter.
Several side-channel attacks have previously been proposed, and one line of research have been attacks against the comparison step of the FO-transform. These attacks construct a decryption failure oracle using a side-channel. A recent work published at TCHES 2022 stresses the need for higher-order masked comparisons by presenting a horizontal attack and proposes a t-probing secure comparison operation. A subsequent work by D'Anvers, Van Beirendonck, and Verbauwhede improves upon the performance of several previous proposals.
In this work, we show that the latter masked comparison suffers from weakness similar to those identified in the former. We first propose an approximate template attack that requires only a very low number of traces for profiling and has an exceptionally high noise tolerance. We show that the profiling phase is not necessary and can be replaced by a vertical analysis of the distribution of certain points of interest without knowledge of the targeted values. Finally, we explain how a horizontal attack may construct a decryption failure oracle from a single trace.
We provide a leakage model of the targeted operations, which is based on the noisy Hamming weight model. Our evaluations are carried out on a physical device to stress the practicality of our attack. In addition, we simulate the attacks to determine the measurement noise levels that can be handled. We discuss the underlying causes for our attack, the difficulty of securing the Fujisaki-Okamoto transform in ML-KEM, and draw conclusion about the (in-)sufficiency of t-probing security in this context.

Keywords: ML-KEM · Kyber · FO-Transform · SCA · Implementation Attack

1 Introduction

The post-quantum algorithms standardization process by the National Institute of Standards and Technology (NIST) recently determined the Key Encapsulation Mechanism (KEM) to be standardized [Natb]. CRYSTALS-Kyber [BDK+18] has been chosen and will be standardized as Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM) [Nata]. ML-KEM bases its security on the hardness of lattice problems. ML-KEM has already been adopted by a number of well-known libraries and applications [Con23, Ehr23, O'B23, Jar22]. The usage on embedded devices is anticipated due to suitable key sizes and beneficial implementation characteristics.

Physical attacks pose a severe challenge to implementations of cryptography on constrained and physically accessible devices. This includes fault injection and side-channel

attacks, which have been widely researched in the context of ML-KEM [RCDB22]. A crucial and often discussed attack vector is the comparison step of the Fujisaki-Okamoto (FO)-transform [GJY19, BDH+21, DHP+22]: ML-KEM defines a Public-Key Encryption (PKE) scheme and uses an FO-transform [FO99, FO13, TU16, HHK17] to obtain a Chosen-Ciphertext Attack (CCA)-secure KEM. This is accomplished, in particular, by decapsulating and decrypting the provided ciphertext in order to return the PKE message. The message is then re-encrypted, and the new ciphertext is compared to the original. If they do not match, the key exchange is rejected implicitly.

If an adversary is able to observe the outcome of this comparison it can be exploited by a CCA to recover the secret [GJY19, BDH+21, DHP+22]: A chosen ciphertext is sent to the KEM and causes a decryption failure which depends on the secret key. If a decryption failure occurs, the re-encrypted ciphertext differs in statistically half the bits. However, if the decryption is successful, the re-encrypted and the submitted ciphertext differ only one bit. If an adversary is able to discriminate between the two cases, a decryption failure inequality can be derived. Multiple such inequalities allow for a recovery of the secret key as discussed in [PP21, HMS+23]. To mitigate this threat the comparison has to be protected against physical attacks.

An often used countermeasure against Side Channel Attacks (SCAs) is masking. Several proposals for masked comparisons exist [OSPG18, BPO+20, BDH+21, CGMZ21, DHP+22, CGMZ23, DBV23]. However, the proposals of [OSPG18] and [BPO+20] are insecure as discussed in [BDH+21]. D'Anvers, Heinz, Pessl, Van Beirendonck, and Verbauwhede [DHP+22] show that the secured variant of [BDH+21] can be targeted by a higher-order horizontal collision attack. While the attack [DHP+22] does not break any security claims, the authors note its practicability and propose a higher order masked comparison which mitigates the attack. A recent work by D'Anvers, Van Beirendonck, and Verbauwhede [DBV23] improves upon the performance of the previous methods including the higher-order masked comparison of [DHP+22].

Our contribution. We show that the proposal of [DBV23] suffers similar weakness as previously identified in [DHP+22]. As in [DHP+22], we do not break any security claims of [DBV23]. However, we show that while the proposed higher-order masked comparison is t-probing secure, its security is questionable in practice.

First, we present a profiled attack that makes use of an approximated template. We show that this attack is very noise tolerant and needs less than 10^3 traces in the profiling phase. The number of traces required in the attack phase is close to the minimum number of traces required in decryption failure attacks (including the previous attack of [DHP+22]) of about 7000 chosen-ciphertext with corresponding traces when targeting ML-KEM768.

Second, we show that the profiling phase is not necessary by using knowledge about the masking scheme and the resulting distributions. The expected distributions can be found by a vertical analysis of the measurements and templates can be built without the knowledge of the target values. Therefore, our vertical attack avoids a profiling phase. The attack performs only marginally worse than the profiled attack, and also requires about 7500 chosen-ciphertexts and corresponding traces for most realistic noise levels.

Third, we propose a horizontal attack that distinguishes between decryption failures and successes using only a single trace. By analyzing the distributions caused by zero and one bits in the masked ciphertext, we may build a template that we subsequently apply to the trace itself. Similar to the vertical attack, the horizontal attack does not require a profiling phase, and achieves comparable success rates. The horizontal attack allows reducing the security of a ML-KEM instance even if only very few traces are available.

To define the attacks, we first provide an analysis of the distributions in the targeted routine, and we build a leakage model based on the noisy Hamming weight (HW) model. We then explain our attacks using the theoretic model, give an intuition on what enables these

attacks, and discuss the t-probing model in this context. To evaluate the proposed approach in a realistic scenario, we capture traces and perform the attacks on a ChipWhisperer UFO board with an `STM32F4` target board [OC14, Inca, Incb]. We complement these results with simulations on multiple noise levels.

Conclusion and implications. We conclude that it seems particularly difficult to secure the comparison step of the FO-transform in ML-KEM. We provide further evidence that proving t-probing security for the FO-comparison in ML-KEM is far from sufficient. Several provable security notions and methodologies have been proposed [BCM+23]. However, to the best of our knowledge, these models have not been evaluated in this context, and have also previously not been used to prove the security of comparison gadgets for ML-KEM. Our work calls for a thorough investigation in protected comparison proposals and for finding better suited models to prove their security.

Limitations and adversarial model. As in all physical attacks, we assume that the attack has physical access to the device under attack; we assume that they can record several thousand of traces. Note, that the previous attack of [DHP+22], motivation for [DHP+22] and [DBV23], requires a comparable amount of traces. In addition, the adversary knows the public key and is able to submit chosen ciphertexts to the device.

Moreover, the attack can be prevented by a countermeasure proposed in [PP21] that locks the device after a certain number of decryption failures. The countermeasure stops all comparable attacks but, unfortunately, allows for Denial of Service (DoS) attacks. The attack that motivated the predecessor [DHP+22] of the masked comparison we target in this work, is also prevented by this countermeasure.

Our attack does not directly apply to the previous comparison methods that use similar methodologies [BDH+21, DHP+22]. This is also true if the instruction set features a Galois field multiplication instruction[1]. Whether our attack applies to the method carried out on such a device depends on the leakage behavior of the used instruction and the precise implementation. As such an implementation is not available to the best of our knowledge, and commonly used instruction sets in embedded devices do not feature such an instruction, we do not answer this question. However, based on our results, such an implementation has to be carefully examined to ensure that our attacks do not apply.

Finally, we carry out our non-profiled attacks assuming that the adversary can select the Point of Interests (POIs) beforehand. We explain how these can be found using only non-profiled techniques as well as by using a profiling device that is used merely for this purpose. Our implementation includes a tool that semi-automatizes this task. However, it still requires manual analysis, and we do not carry out the full non-profiled part of the attack. Nevertheless, using the proposed method, finding points of interest is only a minor obstacle to a determined attacker.

Open source. We publish all resources used for this work. This includes the recorded traces, the leakage simulation, and the implementation of the algorithm for the attacks. These resources are available as auxiliary material to this paper[2].

2 Preliminaries

We first give a high-level overview over ML-KEM, which is the KEM selected for standardization after round 3 of NIST competition [Natb]. In ML-KEM, flipping a single bit of an honestly generated ciphertext may cause the decryption to fail, and if the failing decryption

[1] Such an instruction has been mentioned in [DBV23] as a way to improve the performance.
[2] Available at https://github.com/juliusjh/masked_fo_comparison_attack.

can be observed, an adversary gains information about the long term secret key. Without an implementation attack, the FO-transform [FO99, FO13, TU16, HHK17] prevents observing decryption failures as a chosen ciphertext always leads to a decapsulation error. However, previous attacks have combined chosen-ciphertext attacks with side-channel or fault analysis to observe decryption failures. We explain the underlying principle of these attacks and provide an overview over the literature. Finally, we reiterate a recent proposal for mitigating side-channel attacks on the comparison step of the FO-transform.

Notation. We denote elements of a ring by lower-case letters, vectors by lower-case bold letters, Random Variable (RV) by upper-case letters, and vectors of random variables by bold uppercase letters. For a prime q, we denote the field $\mathbb{Z}/q\mathbb{Z}$ by $\mathbb{F}_q$, and the ring $\mathbb{F}_q[x]/(x^n+1)$ by R_q. The key pair of ML-KEM is denoted by $(\mathtt{pk}, \mathtt{sk})$, ciphertexts by $\mathtt{ct}$, re-encrypted ciphertexts by $\mathtt{ct}'$, manipulated ciphertexts by $\tilde{\mathtt{ct}}$, and the (decrypted) message by $\mathtt{m}$ ($\mathtt{m}'$). Drawing from a random distribution or from a random variable (i.e., from the distribution the random variable follows) is denoted by a "$\xleftarrow{\$}$".

2.1 ML-KEM

ML-KEM is the KEM selected for standardization after the third round of the NIST standardization effort [Natb]. ML-KEM defines a Chosen-Plaintext Attack (CPA)-secure PKE and relies on an FO-transform to obtain a CCA2-secure KEM. ML-KEM performs its operations in the polynomial ring $R_q = \mathbb{F}_q[x]/(x^n+1)$, where $q = 3329$ and $n = 256$.

The PKE consists of the key generation, the encryption, and the decryption algorithm (see Algorithms 1 to 3). The aforementioned CCAs, as well as this work, target the message recovery during the decryption (see Algorithm 2, Line 4). The implementation attacks then target the comparison operation of the decapsulation (see Algorithm 6).

Algorithm 1 PKE.KeyGen

Require: Randomness seeds ρ, σ
Ensure: Public key $\mathtt{pk}$, secret key $\mathtt{sk}$

1: $\hat{\mathbf{A}} \in R_q^{k\times k} \xleftarrow{\$} \text{SampleUniform}(\rho)$
2: $\mathbf{e}, \mathbf{s} \in R_q^k \xleftarrow{\$} \text{SampleBinomial}_{\eta_1}(\sigma)$
3: $\hat{\mathbf{e}}, \hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{e}), \text{NTT}(\mathbf{s})$
4: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{s}} + \hat{\mathbf{e}}$
5: **return** $\mathtt{pk}_{\text{pke}} = (\hat{\mathbf{t}}, \rho), \mathtt{sk}_{\text{pke}} = \hat{\mathbf{s}}$

Algorithm 2 PKE.Dec

Require: $\mathtt{sk} = \hat{\mathbf{s}}$, $\mathtt{ct} = (c_1, c_2)$
Ensure: Decrypted message $\mathtt{m}'$

1: $\mathbf{u} \leftarrow \text{Decompress}(c_1, d_u)$
2: $v \leftarrow \text{Decompress}(c_2, d_v)$
3: $m_{\text{poly}} \leftarrow v - \text{NTT}^{-1}(\hat{\mathbf{s}}^\top \text{NTT}(\mathbf{u}))$
4: $\mathtt{m}' \leftarrow \text{Compress}(m_{\text{poly}}, 1)$
5: **return** $\mathtt{m}'$

Algorithm 3 PKE.Enc

Require: $\mathtt{pk} = (\hat{\mathbf{t}}, \rho)$, $\mathtt{m}$, coins r
Ensure: Ciphertext $\mathtt{ct} = (c_1, c_2)$

1: $\hat{\mathbf{A}} \in R_q^{k\times k} \xleftarrow{\$} \text{SampleUniform}(\rho)$
2: $\mathbf{r}_1 \in R_q^k \xleftarrow{\$} \text{SampleBinomial}_{\eta_1}(r)$
3: $\mathbf{r}, \mathbf{e}_2 \in R_q^k \xleftarrow{\$} \text{SampleBinomial}_{\eta_2}(r)$
4: $e_1 \in R_q \xleftarrow{\$} \text{SampleBinomial}_{\eta_2}(r)$
5: $\hat{\mathbf{r}} \leftarrow \text{NTT}(\mathbf{r})$
6: $\mathbf{u} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}}\hat{\mathbf{r}}) + \mathbf{e}_1$
7: $m_{\text{poly}} \leftarrow \text{Decompress}(\mathtt{m}, 1)$
8: $v \leftarrow \text{NTT}^{-1}(\hat{\mathbf{t}}^\top \hat{\mathbf{r}}) + e_2 + m_{\text{poly}}$
9: $c_1 \leftarrow \text{Compress}(\mathbf{u}, d_u)$
10: $c_2 \leftarrow \text{Compress}(v, d_v)$
11: **return** $\mathtt{ct} = (c_1, c_2)$

Figure 1: The PKE defined by ML-KEM. Note that we simplified the algorithm to provide an overview. For details on parameters and subroutines we refer to [ABD+21]. The relevant locations enabling decryption failure attacks are highlighted.

The **key generation** first samples a random matrix $\mathbf{A} \in R_q^k$, a secret vector $\mathbf{s} \in R_q^k$, and a secret error vector $\mathbf{e} \in R_q^k$, where $k \in \{2, 3, 4\}$. The Module Learning with Errors

Algorithm 4 KEM.KeyGen

Require: Randomness seeds ρ, σ
Ensure: Public key $\mathtt{pk}$, secret key $\mathtt{sk}$

1: $z \xleftarrow{\$}$ SampleUniform()
2: $\mathtt{pk}, \mathtt{sk}_{\text{pke}} \leftarrow$ PKE. KeyGen()
3: $h = \text{H}(\mathtt{pk})$
4: $\mathtt{sk} \leftarrow (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$
5: **return** $\mathtt{pk}, \mathtt{sk}$

Algorithm 5 KEM.Encaps

Require: $\mathtt{pk}$
Ensure: Ciphertext $\mathtt{ct}$, shared secret $\mathtt{K}$

1: $\mathtt{m} \xleftarrow{\$}$ SampleUniform()
2: $\bar{\mathtt{K}}, \mathtt{r} \leftarrow \text{G}((\mathtt{m}, \text{H}(\mathtt{pk}))$
3: $\mathtt{ct} \leftarrow$ PKE. Enc$(\mathtt{pk}, \mathtt{m}, \mathtt{r})$
4: $\mathtt{K} \leftarrow \text{KDF}((\bar{\mathtt{K}}, \text{H}(\mathtt{ct})))$
5: **return** $\mathtt{ct}, \mathtt{K}$

Algorithm 6 KEM.Decaps

Require: $\mathtt{sk} = (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$, $\mathtt{ct}$
Ensure: Shared secret $\mathtt{K}$

1: $\mathtt{m}' \leftarrow$ PKE. Dec$(\mathtt{sk}_{\text{pke}}, \mathtt{ct}_{\text{pke}})$
2: $\bar{\mathtt{K}}', \mathtt{r}' \leftarrow \text{G}((\mathtt{m}', h))$
3: $\mathtt{ct}'' \leftarrow$ PKE. Enc$(\mathtt{pk}, \mathtt{m}', \mathtt{r}')$
4: $b \leftarrow$ Compare$(\mathtt{ct}, \mathtt{ct}')$
5: **if** b **then**
6: **return** $\mathtt{K} = \text{KDF}(\bar{\mathtt{K}}', \text{H}(\mathtt{ct}))$
7: **else**
8: **return** $\mathtt{K} = \text{KDF}(z, \text{H}(\mathtt{ct}))$
9: **end if**

Figure 2: The KEM defined by ML-KEM. Note that we simplified the algorithm to provide an overview. For detailed definitions and parameters we refer to [ABD+21]. The comparison operation and the deterministic encryption step are highlighted. G and H denote hash functions.

(MLWE) sample given by the matrix $\mathbf{A}$ and $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ form the public key, while $\mathbf{s}$ is the secret key.

The **encryption**, creates two additional MLWE samples by sampling $\mathbf{r}_1, \mathbf{r}, \mathbf{e}_1$ and computing $\mathbf{u} = \mathbf{A}\mathbf{r} + \mathbf{e}_1$, and $v = \mathbf{t}^\top \mathbf{r} + e_2$. The message is transformed into a polynomial by mapping zero bits to 0 coefficient and one bits to $\lceil q/2 \rfloor$ coefficients. The ciphertext consists of the compressed versions of $\mathbf{u}$ and $v + m_{\text{poly}}$. The compression and decompression functions are defined as

$$\text{Compress}(x, d) = \left\lceil x \frac{2^d}{q} \right\rfloor \bmod 2^d \tag{1}$$

and

$$\text{Decompress}(x, d) = \left\lceil x \frac{q}{2^d} \right\rfloor. \tag{2}$$

Note that an addition of $\lceil q/4 \rfloor$ in uncompressed form often only causes a single bit to change in compressed form.

The **decryption** first decompresses both ciphertext components and then computes $v - \mathbf{s}^\top \mathbf{u}$ which is equal to

$$m_{\text{poly}} + \mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top (\mathbf{e_1} + \Delta \mathbf{u}) + e_2 + \Delta v, \tag{3}$$

where the Δ terms denote compression errors, m_{poly} the message in polynomial representation and all other terms are called the noise polynomial. Because all the noise terms are small, the message can be recovered with a high probability.

2.1.1 Message Recovery

During the encryption, a 256-bit message $\mathtt{m}$ is encoded into a polynomial in $\mathbb{F}_q[x]/(x^n + 1)$ by mapping a zero bit to a zero coefficient, and mapping a one bit to $\lceil q/2 \rfloor$ (Algorithm 3, Line 7). Using the secret key, the other party may obtain a noisy version of this polynomial,

where the noise on each coefficient depends on the secret key. To recover the message $\mathtt{m}$, a coefficient is mapped to a 0 if and only if it is closer 0 than to $\lceil q/2 \rfloor$ (Algorithm 2, Line 4). If the absolute value of the noise is sufficiently small (i.e., smaller than $\lceil q/4 \rfloor$), the correct bit is recovered. Otherwise, a *decryption failure* occurs, and an incorrect message is returned. The process of first encoding a bit to a coefficient, and then recovering the bit from a noisy coefficient is shown in Figure 3.

2.1.2 The FO-Transform

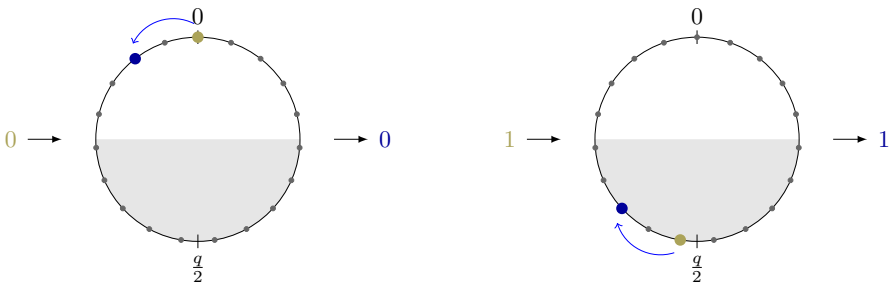
The **key generation** of the KEM first obtains a PKE key pair by invoking the PKE key generation. The KEM public key is then merely the PKE public key, and the secret key is composed of the secret key of the PKE, a hash of the public key, and a random value z.

The **encapsulation** first samples a random message $\mathtt{m}$ and encrypts it into a ciphertext $\mathtt{ct}$ using the encryption routine. This is done deterministically using the randomness derived from $\mathtt{m}$ by hashing. Thus, for each message $\mathtt{m}$ there is a unique[3] valid ciphertext $\mathtt{ct}$; all other ciphertexts are invalid. After the encryption, a shared secret is derived from the ciphertext and the hash of the message from which the randomness was derived.

The **decapsulation** first invokes the decryption routine to obtain $\mathtt{m}'$ using the secret key. This allows one to obtain the same shared secret as the other party. In addition, the encryption routine is invoked to obtain $\mathtt{ct}'$ by encrypting $\mathtt{m}'$. Again, this is done deterministically with the randomness derived from $\mathtt{m}'$ as encapsulation. There is a unique valid ciphertext $\mathtt{ct}$ that corresponds to $\mathtt{m}$, and in case that $\mathtt{ct}$ is not equal to $\mathtt{ct}'$, the decapsulation is implicitly aborted (i.e. the ciphertext is rejected), and z is returned instead of the shared secret. This is called a *decapsulation failure*, which differs from a *decryption failure* (in fact, in this setting, the latter prevents observing the former). The routines of the KEM are shown in Figure 2.

2.2 Decryption Failures and Implementation Attacks

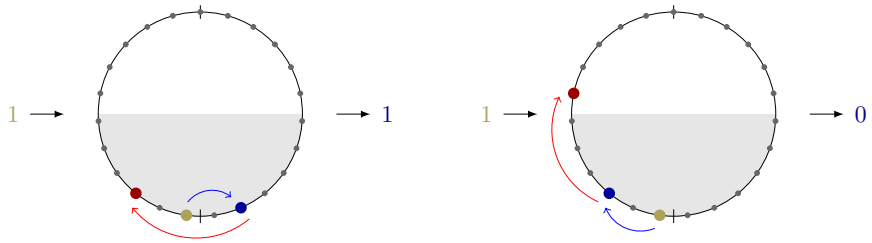
The PKE defined by ML-KEM is not secure against chosen-ciphertext attacks, and the message recovery routine in these class of schemes may be targeted using a chosen ciphertext attack as shown in [Flu16]. An adversary adds $\lceil q/4 \rfloor$ to a single coefficient of an otherwise valid ciphertext. Thereby, a decryption failure occurs if the noise on the coefficient is greater (or greater-equal) than $\lceil q/4 \rfloor$, as shown in Figure 4. If the adversary can generate



(a) 0-bit. (b) 1-bit.

Figure 3: Mapping a bit to a polynomial coefficient in $\mathbb{F}_q$, and then recovering it from a noisy version of the coefficient. A bit is mapped to 0 or $\lceil q/2 \rfloor$; during the decryption, a noisy term (blue error) changes the coefficient, but does not change the recovered bit. Figures adapted from [Her23].

[3] Up to hash collisions.



(a) No failure. (b) Failure.

Figure 4: An error term (e.g., added through a chosen-ciphertext or a fault attack) causes a decryption failure if the noise term is positive. The total noise of a coefficient is given by the sum of the noise term (blue arrow) and the maliciously introduced error (red arrow). Figures adapted from [Her23].

and send such ciphertexts, they would be able to determine whether the noise term on a single coefficient is positive or negative. The noise term itself depends on the secret key, and after several thousands of chosen ciphertexts, the adversary may recover the long term secret key (see, e.g., [HMS+23]).

2.2.1 CCA2-Security

ML-KEM uses a FO-transform to create a CCA2-secure KEM and mitigate the aforementioned attack. The decapsulation algorithm returns a failure whenever a decryption failure happens, regardless of the source of the decryption failure. The comparison of the faulty ciphertext after the re-encryption will fail. This prevents the adversary from gaining any information because decapsulation failures hide decryption failures.

2.2.2 Side-Channel Decryption Failure Oracles

However, a side-channel or fault injection attack may allow adversaries to circumvent this protection. If an adversary can detect the decryption failure by a side-channel the attack can be performed (see, e.g., [GJN20, BDH+21]). This has been exploited in several previous works, e.g., [BDH+21, HPP21, DHP+22, Del22, PP21]. Note that the ciphertexts in question can be chosen such that the added term only changes a single bit in the compressed form (Algorithm 3, Line 10) of the ciphertext.

These attacks construct a decryption failure oracle using an implementation attack. An adversary sends a ciphertext that has been changed by one bit. This manipulated ciphertext will cause a decryption failure whenever the noise term is positive (or zero for 1-bits). If this can be observed using a side-channel, a decryption failure inequality can be derived. Except for [PP21], these attacks work as follows:

1. Honestly generate a valid ciphertext $\mathtt{ct}$.
2. Add $\lceil q/4 \rfloor$ to a coefficient of $\mathtt{ct}$ to obtain $\tilde{\mathtt{ct}}$.
3. Submit $\tilde{\mathtt{ct}}$ to the device; the device computes $\mathtt{ct}'$.
4. Observe whether a decryption failure happens using a physical attack.
5. Derive an inequality over the secret key from the noise term.
6. Repeat from 1. until sufficiently many inequalities have been recorded.
7. Recover the secret key using a key recovery method such as [HMS+23].

Thus, whenever a physical attack reveals whether a chosen ciphertext causes a decryption failure, the secret key of the KEM may be recovered.

Note that the manipulated ciphertext $\tilde{\mathtt{ct}}$ can be chosen to only differ by a single bit from $\mathtt{ct}$. However, if a decryption failure occurs, the re-encrypted ciphertext $\mathtt{ct}'$ will be indistinguishable from a uniformly random ciphertext[4] Therefore, in step 4, the adversary is required to distinguish two cases:

1. The re-encrypted ciphertext $\mathtt{ct}'$ and $\tilde{\mathtt{ct}}$ differ in a single bit[5].
2. The re-encrypted ciphertext $\mathtt{ct}'$ and $\tilde{\mathtt{ct}}$ differ in statistically half of the bits.

2.2.3 Ciphertext Filtering

Pessl and Prokop [PP21] show how the information per used ciphertext can be improved. They suggest a filtering step that removes ciphertexts for which the retrieved inequality holds less information (see also [HMS+23, Section 4.3.1]). The technique amounts to using only ciphertexts that have small Δv (and sometimes e_2) value (see Equation (5)), and it happens without any interaction with the target, i.e., it is done purely offline. The details of this technique are well-known (used, e.g., in [PP21, HPP21, Del22]), and we refer for details to the respective works.

2.2.4 Recovering the Secret Key

In step 5, the adversary obtains a decryption failure inequality, i.e., a linear inequality over the secret key components $\mathbf{x} = (\mathbf{e}, \mathbf{s})$. The noise term in ML-KEM is given by

$$\mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top(\mathbf{e_1} + \Delta\mathbf{u}) + e_2 + \Delta v, \tag{4}$$

where Δu and Δv denote compression artifacts. Therefore, the inequalities obtained by observing decryption failures are of the form

$$(-1)^{\mathtt{obs}}(\mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top(\mathbf{e_1} + \Delta\mathbf{u}) + e_2 + \Delta v)[l] \leq 0, \tag{5}$$

where l is the coefficient in which the error was introduced, and $\mathtt{obs}$ is 0 if no decryption failure was observed and 1 otherwise. Every trace results in an inequality, and the adversary has to recover the secret key from these inequalities.

Several recovery methods for step 7. have previously been proposed [PP21, HPP21, Del22, HMS+23]. The proposal that currently requires the lowest number of traces/faults is presented in [HMS+23]. The recovery methods of [Del22, HMS+23] are error-tolerant, i.e., the key can be recovered if decryptions have been classified incorrectly, and thus some inequalities are incorrect. In fact, the method of [HMS+23] can handle up to 0.4 of the inequalities being incorrect. However, in this case, the required number of inequalities is greatly increased. In this work, we treat the recovery method as a black box and do not improve upon the recovery method.

2.3 Masked Comparisons

The comparison operation of the FO-transform (Algorithm 6, Line 4) has been one of the prime targets of previous discussed CCAs. In fact, [BDH+21] and [DHP+22] show that previous masked implementations were insecure in practice, and propose (higher-order) masked implementations. Additionally, the work of Coron, Gérard, Montoya, Zeitoun [CGMZ21, CGMZ23] and D'Anvers, Van Beirendonck, and Verbauwhede [DBV23] propose more efficient masked comparison operations.

[4] Assuming ML-KEM is secure.
[5] A single coefficient in uncompressed form.

Algorithm 7 Galois field comparison method defined in [DBV23]. The operations are carried out on the individual shares, and $\oplus$, $\odot$ denote addition, Galois field multiplication, respectively.

Require: Boolean shared differences of the ciphertexts $\Delta\mathtt{bc}$.
Ensure: 0 if the difference is a shared zero, otherwise 1.

```
E^(·) ← 0
for all i ∈ {0, ..., length Δbc − 1} do
    r ←$ {0,1}^s
    E^(·) ← E^(·) ⊕ (r ⊙ b[i])
end for
return Or(E^(·))
```

2.3.1 Prior Attacks

The authors of [BDH+21] show that the previous proposals for masked comparisons [OSPG18, BPO+20] leak information using a t-test. Subsequently, this leakage is exploited in the aforementioned manner, and the authors explain how to secure the previously proposed masked comparisons.

D'Anvers, Heinz, Pessl, Van Beirendonck, and Verbauwhede [DHP+22] show that a horizontal collision attack [MME10] on the hash-based comparison of [OSPG18] may be used to construct a decryption failure oracle. While their attack is a second-order attack against a first-order secure implementation, and no security assumptions are broken, the authors stress the practicability of their attack. The comparison of [OSPG18] hashes both submitted and re-encrypted ciphertext, and the authors show that the difference between the trace segments allows distinguishing whether the hash was called on the same or different ciphertexts.

2.3.2 Masked Comparisons

In this work, we investigate the side-channel security of the most recent proposal in [DBV23]. By performing the comparisons in $\mathbb{F}_2^n$, they achieve a performance improvement of roughly 20-25% compared to [CGMZ23] and [DHP+22]. This allows replacing integer multiplications by Exclusive-Ors (XORs) and is coined "Galois Field Compression". The authors prove the security of the proposed gadgets in the t-probing model [ISW03], in particular, the Galois Field Compression is shown [DBV23, Theorem 2] to be t-Strong-Non-Interfering (t-SNI) [BBD+16]. Note that the comparison method has a small but positive probability of collisions, i.e., for invalid ciphertexts to be accepted. The collision probability is 2^{-s} where s is a parameter of the algorithm. The authors claim that this probability cannot be influenced by an adversary.

The core of the Galois Field Comparison algorithm is a subroutine ([DBV23, Algorithm 10, Line 5-7]) that takes the (Boolean) shared ciphertext differences $\Delta\mathbf{u}$ and Δv as inputs, and computes the unshared sum in a secured manner. The differences $\Delta\mathbf{u}$ and Δv are concatenated and treated as a single input of coefficients; we follow their naming scheme in the implementation and denote this vector by $\Delta\mathtt{bc}$[6].

For each shared polynomial coefficient of $\Delta\mathtt{bc}$, a random s-bit value r is sampled. For all bits b of each share, the values of $b \cdot r$ are added up over $\mathbb{F}_{2^n}$, which corresponds to multiplying r and b and XOR'ing the results. If $\Delta\mathtt{bc}$ is a shared zero, the resulting values are all zero. The algorithm is reiterated in Algorithm 7, and the C-code from the implementation associated to [DBV23] is shown in Listing 1.

[6] Implementations often denote $\mathbf{u}$ by $\mathtt{b}$ and v by $\mathtt{c}$.

Listing 1: Bitsliced implementation of Algorithm 7 as proposed by [DBV23]. The outer loop runs over all coefficients of the bitsliced input which we denote by Δbc.

```
for (size_t i = 0; i < SIMPLECOMPBITS; i++)
{
    uint64_t R = random_uint64();
    for (size_t j = 0; j < NSHARES; j++)
    {
        for (size_t k = 0; k < 32; k++)
        {
            biti = (BC_Bitsliced[i][j] >> k) & 1;
            tmp = R * biti;
            E->LSB[j] ^= tmp << k;
            E->MSB[j] ^= tmp >> (64 - k);
        }
    }
}
```

3 Targeting Higher-Order Masked Comparisons

The work of D'Anvers, Heinz, Pessl, Van Beirendonck, and Verbauwhede [DHP+22] stresses the relevance of side-channel security beyond the t-probing model. The authors propose a higher-order masked comparison that improves upon previous work in this regard. Subsequently, D'Anvers, Van Beirendonck, and Verbauwhede [DBV23] improve upon the method in terms of comparison, and prove its security in the t-probing model. We explain how this proposal can again be targeted by attacks that technically do not violate the security guarantees, but have low requirements on the adversary and are likely very simple to carry out in practice. All targeted settings use $t \geq 3$, where t denotes the number of shares and $s = 64$, which is the default of the security parameter in the implementation of [DBV23]; we evaluate our attacks for $t \in \{3, 4\}$.

We first state a high-level overview over the attacks and describe the attacker model. Then, we assume a model for distributions at the targeted locations, and compute relevant properties. Based on the model, we then propose an approximate template attack [CRR02] that requires a low number of traces for profiling. The proposal of [DBV23] does not allow for a straight-forward template attack, as the randomness per share causes the distributions to be non-normal. However, we show that the error introduced by our approximation is rather small; this allows for profiled attacks that have very little requirements on the attacker. We then show that the profiling phase is not required, and the masked comparison allows for a similar attack that obtains a template from attacked traces. The template in our attack is created without knowledge of the targeted values, i.e., no profiling phase is required. Instead, we create the template without the knowledge of labels from traces that are subsequently also used to recover the targeted values. Finally, we propose a horizontal attack that creates a template from a single trace – again, without a profiling phase. By estimating the distributions for the targeted values from several locations in the trace, we may recover the shares without profiling.

Note that the attacks in this section are entirely based on the model assumed in Section 3.1.3. From this model, the analysis in Section 3.2 follows, which be base our attacks on. We show that our model is realistic in the next section where we target a physical device.

3.1 Adversarial Model

The attacker model is similar to the attacks of [BDH+21] and [DHP+22], but targets a different, higher-order protected implementation. In addition, we rely on the recovery

Listing 2: The targeted inner loop of Listing 1 as in the implementation of [DBV23].

```
biti = (BC_Bitsliced[i][j] >> k) & 1;
tmp = R * biti;
E->LSB[j] ^= tmp << k;
E->MSB[j] ^= tmp >> (64 - k);
```

method proposed in [HMS+23] and ciphertext filtering as described in [PP21].

3.1.1 High-Level Overview

The basis for the attacker model is the strategy also used in [BDH+21, DHP+22], reiterated in Section 2.2. This means, the adversary creates m chosen-ciphertexts $\tilde{\mathtt{ct}}$ that differ in a single coefficient (or even only a single bit) from an honestly generated ciphertext $\mathtt{ct}$. The ciphertext should be filtered according to the method proposed in [PP21] (see also [HPP21, HMS+23]). These ciphertexts are sent to the device under attack using the method of [DBV23] and cause a decryption failure in statistically half of the cases (and decapsulation failures in all cases). The adversary then records power traces of the "Galois Field Compression" routine (see [DBV23, Algorithm 10], reiterated in Algorithm 7 and Listing 1). From the power traces, they are asked to distinguish between decryption failures and decryption successes. If this is successful with an accuracy of more or equal to 0.8, the secret key can be recovered.

For simplicity, we explain our attacks for a single coefficient of $\Delta\mathtt{bc}$ (see Section 2.3.2). The extension to several coefficients of $\Delta\mathtt{bc}$ is straight-forward. Note that the number of traces m has to be chosen depending on the capabilities of the attacker to run a subsequent lattice reduction, and the accuracy of the classification (see [HMS+23]). In Section 4, we provide an exact number for an attack on a physical device.

3.1.2 Targeted Routine

We target the core routine of the comparison proposed by [DBV23] (see Listing 1). The input to the routine is $\Delta\mathtt{bc}$, a t shared vector of length 272 of 32-bit integers, and we aim at recovering as many bits of each share as possible. However, it is sufficient to find a single unshared bit that does not XOR to zero, which is the case in statistically half the bits if a decryption failure occurs. Thus, recovering the bits with low accuracy is not a problem, as the information is encoded in $271 \cdot 32$ shared bits. Listing 2 shows the targeted inner loop of Listing 1; we aim at recovering the value of `biti`.

3.1.3 Leakage Assumption and Notation

In the following, we assume the HW leakage model, and target the inner loop in Listing 2. Let l denote the index of a decapsulation/trace, i denote the index of the current coefficient of $\Delta\mathtt{bc}$, j runs over the share indices, and $k \in \{0, \ldots, 32\}$ loops over the bits. The values $b_{l,i,j,k}$ and $r_{l,i}$ model the bit extracted in the inner loop and the randomness per element of $\Delta\mathtt{bc}$.

Following the model, we assume that the inner loop of Listing 2 leaks the multiplication of $b_{i,l,j,k}$ and $r_{l,i}$. Let $h_{l,i} = \mathrm{HW}(r_{l,i})$ be the HW of $r_{l,i}$, then we assume to observe

$$\mathtt{obs}_{l,j,k} \xleftarrow{\$} \mathcal{N}(b_{i,l,j,k} \cdot h_{l,i}, \sigma), \tag{6}$$

where $\mathcal{N}$ denotes a normal distribution and σ is the standard deviation of the measurement noise. For simplification, we assume that we only have a single POIs per bit. Therefore, POIs can be indexed by triples (i, j, k) and correspond to a location in the traces.

3.2 Leakage Distributions

A straight-forward template attack on the inner loop would be to record templates for all $s+1$ possible HWs of r. However, recording s templates, where s is proposed to be 32 or 64, is hard and not necessary. We show that an adversary may record an approximate template from at a POI ignoring that r have in fact different HWs. Let in the following $\texttt{poi} = (i, j, k)$ be a POI and let $\mathcal{L}_{\texttt{poi},b}$ be the indices of the traces where $b_{l,i,j,k} = b$.

3.2.1 Distributions at 1-Bits

Given $b_{l,i,j,k} = 1$, the observed value at $\texttt{poi}$ in a trace l is

$$\texttt{obs}_{l,\texttt{poi}} \xleftarrow{\$} \mathcal{N}(h_{l,i}, \sigma). \tag{7}$$

Let $\texttt{Obs}_{\texttt{poi}}$ be the RV of randomly choosing a trace l and selecting $\texttt{obs}_{l,\texttt{poi}}$. This means, for $\texttt{Obs}_{l,\texttt{poi}}$ denoting the RVs for the observations at a single trace, $\texttt{obs}_{\texttt{poi}}$ is given as

$$\texttt{obs}_{\texttt{poi}} \xleftarrow{\$} \texttt{Obs}_{l,\texttt{poi}}, \tag{8}$$

where $l \xleftarrow{\$} \mathcal{L}_{\texttt{poi},1}$. This is equivalent to

$$r_{l,i} \xleftarrow{\$} \{0,1\}^s \tag{9}$$

$$\texttt{obs}_{\texttt{poi}} \xleftarrow{\$} \mathcal{N}(\mathrm{HW}(r_{l,i}), \sigma). \tag{10}$$

Thus, $\texttt{Obs}_{\texttt{poi}}$ is distributed as mixture distribution of weighted normal distributions:

$$\sum_{h \in \{0,\ldots,s\}} P(h)\mathcal{N}(h, \sigma), \tag{11}$$

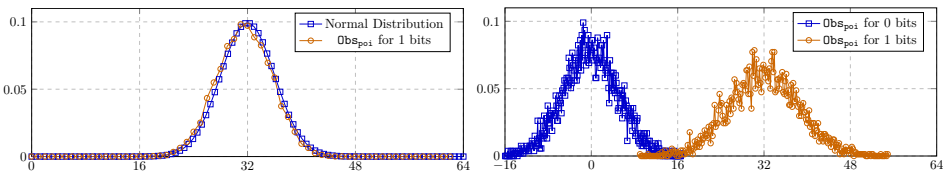
where the distribution of the HWs is the sum of RVs following a Bernoulli distribution and thus follows a binomial distribution with $\eta = s$. The distribution of $\texttt{Obs}_{\texttt{poi}}$ is shown in Figure 5.

3.2.2 Distributions at 0-Bits

Under the leakage assumption, for $l \in \mathcal{L}_{\texttt{poi},b}$, the distributions of $\texttt{Obs}_{l,\texttt{poi}}$ are all distributed according to

$$\texttt{obs}_{l,\texttt{poi}} \xleftarrow{\$} \mathcal{N}(0, \sigma). \tag{12}$$

Thus, in this case, $\texttt{Obs}_{\texttt{poi}}$ is normally distribution around 0 with standard deviation σ.



(a) $\texttt{Obs}_{\texttt{poi}}$ for 1-bits. (b) $\texttt{Obs}_{\texttt{poi}}$ for 0-bits and 1-bits.

Figure 5: The distribution of $\texttt{Obs}_{\texttt{poi}}$ for 1-bits simulated for 7000 traces and a normal distribution with the same mean and variance (Figure 5a), and the distributions for both bits (Figure 5b) simulated according to our model (with $\sigma = 5.0$).

3.3 Profiled and Non-Profiled Attacks

We are now ready to describe several attacks based on the distribution assumptions provided in the previous section. While these are only assumptions, and our attacks are based on them, we demonstrate in the following section that they accurately resemble the leakage behavior of a physical device.

3.3.1 Approximate Template Attacks

To build a template, we would need to model the mixture distribution of $\texttt{Obs}_{\texttt{poi}}$ given recorded traces. Given a large amount of samples, it could be possible to model these distributions directly. However, as we only require distinguishing between a normal distribution and the mixture distribution, we can make use of a much simpler option: Ignoring that $\texttt{Obs}_{\texttt{poi}}$ for 1-bits is the mixture of several normal distributions with different means, we may also compute a template assuming that the mixture distribution is normal. If the noise level is sufficiently high and there is a large amount of samples, the mixture distribution is very close to a normal distribution because the factors follow a binomial distribution (see Equation (11) and Figure 5b). Thus, as we need to record several thousand traces in every case to obtain sufficiently many decryption failures/successes, the error we make when applying the template will be small.

3.3.2 Vertical Non-Profiled Attacks

In a profiled template attack, we may compute templates for 0- and 1- bits from known $\Delta\texttt{bc}$ values. However, under our assumptions, the distributions for 0- and 1- bits have different means and only partially overlap for even for very high noise-levels (see Figure 5b). In fact, we may categorize the observations at all POIs belonging to bit into two sets (potentially with non-empty intersection). From these sets for 0- and 1- bits, we can compute the template just as described in the previous section.

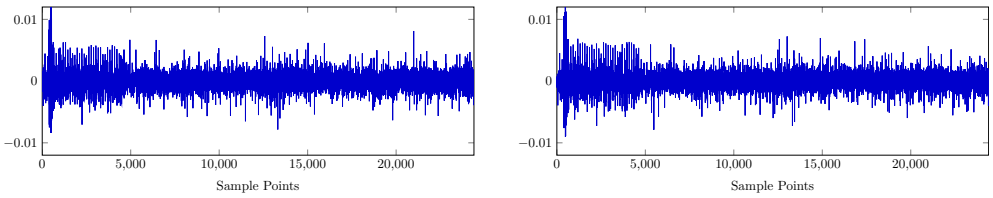
Note that separating the distributions in the simplest case is a merely using the mean of the distribution as a threshold. However, if the distributions overlap, computing the mean and the variance can be done only on the non-overlapping parts of the distributions, and allows for more precise template. In our attack on a physical device described in Section 4, the distributions could usually be separated with only little errors. Applying the template then amounts to computing the probability of a single observation belonging to a 0 or a 1, based on this separation.

To entirely eliminate the profiling phase, POIs have to be found without knowledge of the $\Delta\texttt{bc}$ as well. We explain how this can be achieved by manual analysis in Section 3.3.4.

3.3.3 Horizontal Non-Profiled Attacks

In the attacker model, an adversary is required to record a large number of traces as only a single inequality can be obtained per chosen-ciphertext. Therefore, it is no limitation that vertical non-profiled require a larger number of traces to model mixture distributions correctly. Nevertheless, Hermelink et al. [HMS+23] show that even recovering only a few inequalities, i.e., using only a few traces in our setting, already suffices to reduce the security of ML-KEM instance. In this setting, there might not be sufficiently many traces for building a template without knowledge of the $\Delta\texttt{bc}$. Moreover, the targeted higher-order masked proposals were motivated by the possibility of horizontal higher-order side-channel attacks that were comparably simple to carry out in practice. Thus, we also propose a horizontal higher-order attack.

In fact, for a single coefficient of $\Delta\texttt{bc}$, the same random value r is used. For t shares, we may target $t \cdot 32$ computations of the inner loop. As in vertical non-profiled attacks, we



(a) Unprofiled (b) Profiled

Figure 6: Difference of means after separating distribution a correctly guessed POI for the first bit of a share, and difference of means after separating by known $\Delta\mathtt{bc}$. Both plots were obtained in the setting of Section 4 with 3 shares.

may then separate the distributions and compute a template without knowledge of $\Delta\mathtt{bc}$. However, in this case, both distribution are Gaussian.

3.3.4 Finding Points of Interest

The vertical and horizontal non-profiled attacks require the knowledge of POIs per targeted bit. In a non-profiled setting, finding these locations can be achieved by manual analysis as follows. For each potentially interesting location, the adversary assumes that the location is distributed according the model we presented in this section. Under this assumption, they may separate the traces in two classes, and abort if one class contains significantly more samples than the other. If both sets contain a similar amount of samples, they then compute the mean for both classes. If the locations is a POI for the first bit of a share, the resulting plot will contain one large spike as well as 32 evenly spaces smaller spikes; the latter comes from the accumulating variable in the inner loop. This pattern, shown in Figure 7b, allows distinguishing the first POI of a share from other locations in the traces. The remaining POIs for the share lie in between the evenly spaced spikes, and show a large spike at a single location. Figure 6 shows the results of our method and the POIs found in a profiled setting.

Alternatively, a profiling device of a similar kind to the targeted device may be used. However, as opposed to common profiled attacks, the profiling device is only used to find POIs. Moreover, a pure brute-force attack is also an option, even though it requires some computational effort.

3.4 Underlying Principle and t-Probing Security

The proposal of [DBV23] is shown to be t-probing secure, and we do not break the security guarantees. However, the practical complexity of our attacks is not increased by masking – in fact, higher masking order may improve upon the attacks as more samples are available. This leads to the question of what underlying principle enables these attacks, and why the t-probing security is not sufficient in the case of FO-transform comparisons.

3.4.1 Information Learned from FO-Comparisons

When targeting the FO-comparison, the adversary aims to learn a single bit – decryption failure or decryption success. This bit is encoded in all coefficients of the difference of ciphertext $\Delta\mathtt{bc}$ with very high probability: If a decryption failure occurs, the re-encrypted ciphertext cannot be distinguished from randomness, and the probability of having a coefficient being equal to the submitted ciphertext is very low. Thus, in fact, even a single bit of $\Delta\mathtt{bc}$ allows detecting a decryption failure with probability $1/2$.

This means that a target bit is encoded in a large number of processed values. Therefore, attacks require less accuracy, and horizontal and template attacks are easier to carry out.

But most importantly, it enables analyzing the distributions at POIs without the knowledge of actual labels. As a result, the masked FO-comparisons is particularly vulnerable to higher-order attacks. It is crucial to ensure that (bits of) coefficients of $\Delta\mathtt{bc}$ cannot be distinguished by analyzing the distributions even with a large amount of samples.

3.4.2 The Masked Comparison of [DBV23]

In comparison to the previous proposals of [BDH+21, DHP+22], the proposal of [DBV23] processes bits of $\Delta\mathtt{bc}$ individually. Every bit of a coefficient of $\Delta\mathtt{bc}$ are polynomial coefficients over $\mathbb{F}_2$. While working over a Galois field enhances performance, it also causes the zero and one bit distributions to be clearly observable. This greatly reduces the effort for template attacks, and it enables the vertical non-profiled attack we presented. The large amount of sample per trace with the same random mask enables our horizontal non-profiled attack. In fact, a larger amount of shares improves our attack – higher-order masking in the proposed way **reduces** the security of the comparison against our attacks.

4 Validating the Model

The attacks that we described in the previous section are based on the assumptions made in Section 3.1.3 and on the noisy HW model. It is necessary to demonstrate that these assumptions hold in practice and model a physical device sufficiently well to execute the proposed attacks. We are using the ChipWhisperer [OC14] for our practical evaluations.

4.1 Measurement Setup

We target the open-source implementation[7] of [DBV23] using a ChipWhisperer UFO board with an `STM32F4` target board [OC14, Inca, Incb]. Each trace has 24400 sample points and the targeted implementation was compiled with `-O2`[8]. To obtain the values for $\Delta\mathtt{bc}$ that are used as labels during profiling, we ran the comparison method on x86 with the same fixed randomness. Note that we identified undefined behavior in the comparison method (bitshift by integer width), which we corrected for the x86 implementation. We analyzed the traces and recover the values of coefficients of $\Delta\mathtt{bc}$ using a Python implementation.

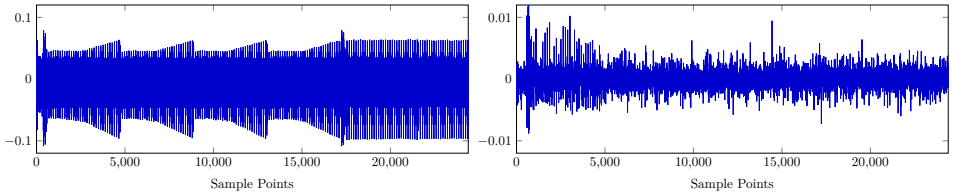
For the following evaluation, we first create a valid ciphertext pair $\mathtt{ct} = (\mathbf{u}, v)$ and create a copy $\mathtt{ct}' = (\mathbf{u}', v')$ as re-encrypted ciphertext. Then, we add an error to v to simulate the chosen ciphertext that is sent to the device. To simulate decryption successes, we collect a trace for the comparison method called with these to inputs. To simulate decryption failures, we compute a new, random ciphertext pair $\tilde{\mathtt{ct}}$ and call the comparison on $\mathtt{ct}$ and $\tilde{\mathtt{ct}}$.

4.2 Analyzing the Distributions

We first take a look at the reality of distributions that were modeled in Section 3. To achieve this, we record 500 traces for decryption failures and 500 traces for decryption successes. While this is less than required to carry out the full attack (as every trace gives at most one decryption failure inequality), it in fact suffices to distinguish almost all traces from just the first coefficients of $\Delta\mathtt{bc}$.

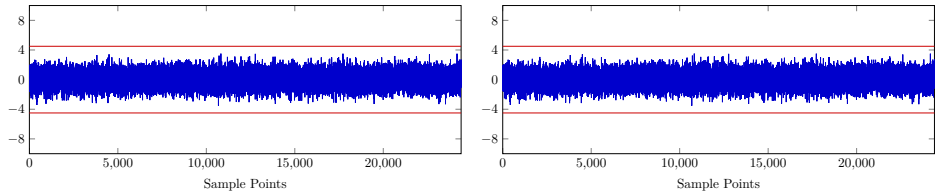
From the mean of the recorded distributions, inner loops and the individual shares can be clearly identified, as shown in Figure 7a. The difference in means for 0 and 1 bits is shown in Figure 7b, and it can be seen that the POIs for later bits are correlated to the POIs for the first bit. This is likely the result of the accumulating variable in Listing 2.

[7] Published at https://github.com/KULeuven-COSIC/Revisiting-Masked-Comparison/.
[8] The results for compilation with `-O3` differ only marginally.



(a) Mean trace. (b) Difference of means.

Figure 7: The mean trace and the difference of means depending on the first bit of the first coefficient of $\Delta\mathtt{bc}$. The figures were obtained using 1000 traces and with 4 shares, and they show the processing of the first coefficient and half of the second coefficient of $\Delta\mathtt{bc}$.



(a) Three shares. (b) Four shares.

Figure 8: The t-test statistics for three and four shares. The bound of 4.5 is shown in red.

4.2.1 Welsh's t-Test

In order to ensure that our attacks are not only based on easily detectable implementation errors, we conduct a t-test. We use the same traces for the t-test as for our later evaluation. The traces were grouped by decryption failure and success and the t-values were calculated. The results are presented in Figure 8 and show no significant first-order leakage. Note that in our sketched attack path we do not directly target the information of success and failure. Instead, we partially recover the shares of $\Delta\mathtt{bc}$.

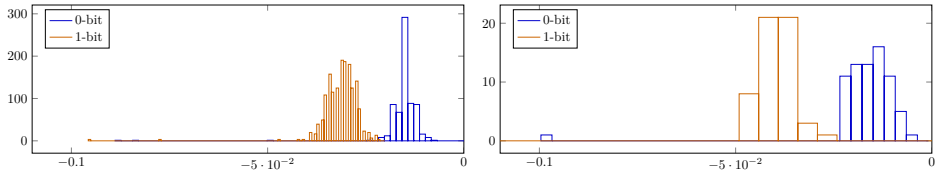
4.2.2 Distributions at 0 and 1 Bits

The vertical and horizontal distribution modeled in Section 3.2 are shown in Figure 9a and Figure 9b, respectively. The distinct distributions required by our model are easily identifiable and separating the distributions can be done with just a few errors. Only a few outliers cannot be clearly attributed to one of the distributions. Clearly, several POIs per bit of each coefficient of $\Delta\mathtt{bc}$ give more reliable templates in our vertical attacks. We do not display multivariate distributions in this part for simplicity, but instead report on the results for several POIs in the following section.

4.3 Carrying out the attacks

For evaluation, we again used 1000 traces of which 500 are decryption failures and 500 are decryption successes. Note that an attacker can at most obtain 1000 decryption failure inequalities from 1000 traces, but requires at least 5500 to recover the secret key directly. Nevertheless, we chose to use fewer traces to take different settings into account in which only a lower number of traces is available.

We target the first coefficient of $\Delta\mathtt{bc}$, this means that 270 other coefficient allow improving the precision of our attack. We aim at recovering the shares of the coefficient of $\Delta\mathtt{bc}$; in a decryption success, these should all XOR to zero, while in a failure, this should be the case in statistically half the bits. As we cannot recover each bit without errors, we



(a) Vertical distribution. (b) Horizontal distribution.

Figure 9: Vertical distribution and horizontal distributions. The vertical distribution shows the distribution at the first POI of the first bit of the first coefficient of the first share of $\Delta\mathtt{bc}$, and the horizontal distribution shows the distribution at POIs for the first coefficient of $\Delta\mathtt{bc}$ over all shares and bits.

Table 1: Results for profiled and non-profiled attacks on 3 and 4 shares.

Attack	Shares	Traces (Profil/Attack)	Classified	Correct
Template	3	500/500	0.84	1.00
Vertical	3	0/1000	0.84	1.00
Horizontal	3	0/1000	0.82	1.00
Template	4	500/500	0.87	1.00[9]
Vertical	4	0/1000	0.85	1.00
Horizontal	4	0/1000	0.85	1.00[9]

classify by the amount of bits summing to zero – if less than 0.55 of the bits do not sum to zero, we assume a decryption failure, and if more than 0.8 xor to zero, we assume a decryption success. For traces where classification between decryption failure and success is unclear, we may simply use more coefficients of $\Delta\mathtt{bc}$. Note that the recovery method of [HMS+23] (as well as the recovery method of [Del22]) allows for recovery even in the presence of incorrectly classified decryption successes/failures.

To evaluate the template attack, we first record a template using merely 500 traces and evaluate the performance on the remaining 500 traces. Both vertical and horizontal non-profiled attacks are carried out on all 1000 traces. Note that we use only 1/271 of the available information; the success and classification ratios can be increased by using more coefficients of $\Delta\mathtt{bc}$ and stricter bounds.

The results for 3 and 4 shares are shown in Table 1. Note that more shares slightly improve the horizontal attack; this might seem counter-intuitive. However, in our setting, more shares give us more samples to build our template. Thus, this is to be expected.

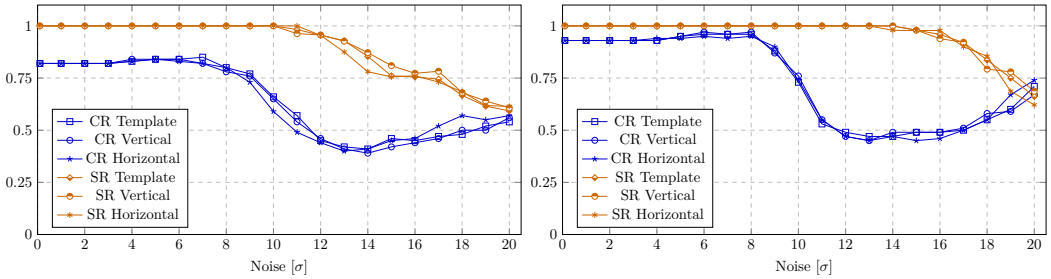
5 Evaluation

In addition to the attacks on a physical device described in the previous section, we evaluate our attacks using simulations using the model described in Section 3.

5.1 Simulation

We implemented the model described in Section 3 in Python. We simulate the attack in dependence to the number of coefficients of $\Delta\mathtt{bc}$, the number of shares, the number of POIs per targeted bit, and the number of traces. The noise level is given as standard deviation σ of the used noisy HW model. Given these parameters, we sample simulated traces, by sampling each POI from the RV defined in Section 3. These simulated traces

[9] Note that one trace was classified incorrectly.



(a) Targeting one coefficient.

(b) Targeting five coefficients.

Figure 10: Results in terms of classification and success rate per standard deviation σ for one and five coefficients of $\Delta\mathtt{bc}$. We also state some of the results in Table 2.

are used to carry out the same attack as on the physical device's traces. Note that the attacks on the physical device contains some outliers (see Figures 9a and 9b) and that these are not included in our simulation.

5.2 Results

In the following, we report on the results using our simulation. We evaluate different settings on 7000 traces for varying noise levels, targeting 1 and 5 coefficients of $\Delta\mathtt{bc}$, and with 4 shares.

5.2.1 Distinguishing Decryption Failures and Successes

We first report on the results in terms of distinguishing between decryption failures and successes. Note that the full attack does not require us to classify all traces and only requires a success rate of more than 0.8. Lower success and classification rates merely increase the required number of traces. We report on the required number of traces for the full attack in the next section.

All results are stated for standard deviation $\sigma \in \{0.1, 1.0, 2.0, \ldots, 20.0\}$. In Figure 10a, we report on the success and classification rates for targeting 5 coefficients of $\Delta\mathtt{bc}$, and Figure 10b shows the setting in which we target 5 coefficients of $\Delta\mathtt{bc}$.

5.2.2 Full Attack using [HMS+23]

From classifying the recorded traces, we may record decryption failure inequalities and recover the secret key using the method of [HMS+23]. We use the implementation[10] provided with [HMS+23], but modify it to work with the security level 768. Each classified trace results in a decryption failure inequality. While unclassified traces simply do not contribute, incorrectly classified inequalities worsen the success rate. We estimate the number of required traces using the implementation of [HMS+23], with ciphertext filtering with `max-delta-v=5`. The number of required traces to recover the secret key for security level 768 in the setting of Figure 10b is shown in Table 2. The minimum number of required traces in attacks exploiting decryption failures is stated under the assumption that an attacker uses the same recovery method.

To obtain the total number of traces, we first evaluate the number of required traces for the given correctness probability in [HMS+23]. In case of a correctness probability of 1 (corresponding to a success rate of 1) and ML-KEM768, about 7000 traces are required. From this number, the total number of traces can be determined. Note that in many cases

[10] Available at https://github.com/juliusjh/improved_decryption_error_recovery.

Table 2: Approximate number of required traces to recover an ML-KEM 768 key per standard deviation σ targeting 5 coefficients of $\Delta\mathtt{bc}$ in the horizontal attack with 4 shares; evaluated on 100 traces. Number of traces rounded to the nearest multiple of 500. The minimum number of traces for success and classification rate 1 is 7000.

σ	≤ 2	5	10	15	20
Classification Rate	0.93	0.94	0.74	0.45	0.74
Success Rate	1.00	1.00	1.00	0.98	0.62
Required Traces	7500	7500	9500	18000	–

the key can already be recovered with significantly fewer traces. For example, with 6500 traces, we already observe a success rate of 0.9 and remaining BIKZ[11] of 140. Moreover, even with only a few traces, the security of the instance of the scheme is already reduced.

6 Conclusion

We built a leakage model for the masked comparison proposal of [DBV23], and proposed several attacks based on it. The profiled attack requires only a very small number of traces for profiling, and both profiled and non-profiled attacks have exceptionally high noise tolerance. While we do not break any security claims of [DBV23], we conclude that the masked comparison is highly insecure against side-channel attacks in practice. Our attacks further stress the need and the difficulty of securing the FO-transform in ML-KEM against attacks exploiting decryption failures.

6.1 Countermeasures

Several countermeasures could mitigate our attack. However, all of them have severe drawbacks – either they enable different types of attacks, or they are costly in terms of performance. Therefore, we recommend to evaluate the security of previous proposals [DHP+22], and to use those instead. Note that similar attacks may apply against these proposal as well. However, our exact attacks do not directly apply as these proposals do not work over $\mathbb{F}_2$, i.e., on single bits of coefficients of $\Delta\mathtt{bc}$.

6.1.1 Shuffling Countermeasures

Shuffling could be applied at three different levels: If the complete loop, including the random constant is shuffled, the latter has to be stored and loaded from memory. This is costly, and the HW of the constant can likely be detected which would allow for an approximate un-shuffling again. Shuffling the inner loop (processing bits) only worsens our success rate, but probably does not prevent the attack. Instead of recovering the shares of a single coefficient of $\Delta\mathtt{bc}$, the adversary can search for a coefficient of $\Delta\mathtt{bc}$ in which the HW of the individual share do not match those of a masked zeros. Finally, shuffling share indices as well as the inner loop could mitigate the attack. While we can still recover the bits of the shares, we may not assign them to shares or positions. Nevertheless, we see this as risky as even the total number 1-bits leaks some information: In a decryption success, it must be divisible by two. Moreover, shuffling has previously been reversed using a wide-variety of techniques (see, e.g., [TH08, RPD09, VMKS12, BGNT18, BS20, ABG+22, HSST23, BNGD23]). In addition, the attacker is now only required to (partially) recover the shuffling permutation, which is a much simpler task than targeting a more secure and shuffled implementation. Further, it is questionable if these additional measures do not eat

[11] BKZ-β required to obtain the secret key from the remaining lattice instance.

up the claimed performance improve of [DBV23]. For these reasons, we did not attempt to fix the proposed method using shuffling.

6.1.2 Dummy Operations

Carefully introduced dummy operations that perform comparisons on random data could mitigate our attacks. If the attacks are prevented depends on the exact way this is carried out. However, inserting dummy operations most likely negates the performance gains over [DHP+22], too.

6.1.3 The Countermeasure of Pessl and Prokop

Pessl and Prokop [PP21] suggest shutting down the device after a certain number of decryption failures have occurred. This countermeasure fully prevents our attacks as well as all previous attacks that exploit decryption failures. However, Pessl and Prokop also note that it also leaves the device vulnerable to a very simple DoS attack.

6.2 Future Work

The comparison of the FO-transform is a highly sensitive operation of ML-KEM and several other post-quantum KEMs. Whenever the comparison operation leaks information about whether a decryption failure occurred, the secret key can be recovered in a few thousand traces. In this work, we only targeted the latest and most performant masked comparison proposal of [DBV23]. Evaluating the security of other masked comparison methods in similar manners is highly relevant for the wide-spread adoption of ML-KEM in the embedded world.

6.2.1 Different Masked Comparisons

The masked comparison of [DBV23] replaced the floating point arithmetic of [DHP+22] by Galois field arithmetic, i.e, sees the bits of a value as coefficient of a polynomial over $\mathbb{F}_2$. Targeting the comparison of [DHP+22] requires working on HWs of the product of random constant and the coefficient of $\Delta\mathtt{bc}$ (instead of a single bit of the coefficient). Understanding under which circumstances and with what noise tolerance such attacks can be carried out is an open question.

6.2.2 Galois Field Multiplication in Hardware

The investigated and exploited leakage model may not be observable in certain processor architectures. This might be the case, for instance, if there is an explicit command for Galois field multiplication in the instruction set or if the noise levels are even greater. In general, the security of the comparison method depends on the leakage behavior of the Galois field instruction and its application. A meticulous evaluation is necessary to make sure that attacks of this kind are mitigated.

6.2.3 Modeling the Comparison

In ML-KEM, a chosen-ciphertext that differs by a single bit may cause a decryption failure that leaks information if it can be observed. This makes the FO-transform comparison step particularly difficult to defend. The notion of t-probing security has been known to not necessarily fit the reality of a physical device [BCPZ16], but is nevertheless widely used. However, different proposals exist [BCM+23] and could help securing the FO-transform in ML-KEM. We provide further evidence for the necessity of advanced models to prove security against physical attacks.

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation (version 3.02), 2021.

[ABG+22] Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos, and François-Xavier Standaert. Bitslice masking and improved shuffling: How and when to mix them in software? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):140–165, 2022.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BCM+23] Sonia Belaïd, Gaëtan Cassiers, Camille Mutschler, Matthieu Rivain, Thomas Roche, François-Xavier Standaert, and Abdul Rahman Taleb. Towards achieving provable side-channel security in practice. *IACR Cryptol. ePrint Arch.*, page 1198, 2023.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison for lattice-based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):334–359, 2021.

[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 *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26 , 2018*, pages 353–367. IEEE, 2018.

[BGNT18] Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, and Yannick Teglia. Multivariate high-order attacks of shuffled tables recomputation. *J. Cryptol.*, 31(2):351–393, 2018.

[BNGD23] Linus Backlund, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Secret key recovery attack on masked and shuffled implementations of crystals-kyber and saber. In *Applied Cryptography and Network Security Workshops - ACNS 2023 Satellite Workshops, ADSC, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Kyoto, Japan, June 19-22, 2023, Proceedings*, volume 13907 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2023.

[BPO+20] Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):483–507, 2020.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[CGMZ21] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR Cryptol. ePrint Arch.*, page 1615, 2021.

[CGMZ23] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):153–192, 2023.

[Con23] Contributors to libsignal. Release v0.27.0, 2023. https://github.com/signalapp/libsignal/releases/tag/v0.27.0.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DBV23] Jan-Pieter D'Anvers, Michiel Van Beirendonck, and Ingrid Verbauwhede. Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations. *IEEE Trans. Computers*, 72(2):321–332, 2023.

[Del22] Jeroen Delvaux. Roulette: A diverse family of feasible fault attacks on masked kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):637–660, 2022.

[DHP+22] Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. Higher-order masked ciphertext comparison for lattice-based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):115–139, 2022.

[Ehr23] Ehren Kret. Quantum resistance and the signal protocol, 2023. https://signal.org/blog/pqxdh/.

[Flu16] Scott R. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. *IACR Cryptol. ePrint Arch.*, page 85, 2016.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *LNCS*, pages 537–554. Springer, 1999.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptol.*, 26(1):80–101, 2013.

[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, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 359–386. Springer, 2020.

[GJY19] Qian Guo, Thomas Johansson, and Jing Yang. A novel CCA attack using decryption errors against LAC. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 82–111. Springer, 2019.

[Her23] Julius Hermelink. Decryption errors and implementation attacks on kyber, 4 2023. Talk at the Institute für IT-Sicherheit at Universität zu Lübeck.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

[HMS+23] Julius Hermelink, Erik Mårtensson, Simona Samardjiska, Peter Pessl, and Gabi Dreo Rodosek. Belief propagation meets lattice reduction: Security estimates for error-tolerant key recovery from decryption errors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):287–317, 2023.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-Enabled Chosen-Ciphertext Attacks on Kyber. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15 , 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 311–334. Springer, 2021.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of ntts. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):60–88, 2023.

[Inca] NewAE Technology Inc. Cw308 ufo. https://rtfm.newae.com/Targets/CW308%20UFO/.

[Incb] NewAE Technology Inc. Cw308 ufo stm32f4. https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F/.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[Jar22] Brian Jarvis. How to tune tls for hybrid post-quantum cryptography with Kyber, 2022. https://aws.amazon.com/blogs/security/how-to-tune-tls-for-hybrid-post-quantum-cryptography-with-kyber.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2010.

[Nata] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. https://csrc.nist.gov/pubs/fips/203/ipd.

[Natb] National Institute of Standards and Technology. PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4.

[O'B23] Devon O'Brien. Protecting chrome traffic with hybrid kyber kem, 2023. https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Paper 2014/204, 2014. https://eprint.iacr.org/2014/204.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on cca-secure lattice kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):37–60, 2021.

[RCDB22] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. Cryptology ePrint Archive, Paper 2022/737, 2022. https://eprint.iacr.org/2022/737.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[TH08] Stefan Tillich and Christoph Herbst. Attacking state-of-the-art software countermeasures-a case study for AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2008.

[TU16] Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the fujisaki-okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216, 2016.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.