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

Julius Hermelink[1], Kai-Chun Ning[1], Richard Petri[1] and Emanuele Strieder[2,3]

[1] MPI-SP, Bochum, Germany
julius.hermelink@mpi-sp.org
kai-chun.ning@mpi-sp.org
rp@rpls.de
[2] Fraunhofer AISEC, Garching, Germany
emanuele.strieder@aisec.fraunhofer.de
[3] Technical University of Munich, Germany

Abstract. NIST released the draft standard for ML-KEM, and we can expect its widespread use in the embedded world in the near future. Several side-channel attacks have been proposed, and one line of research has focused on attacks against the comparison step of the FO-transform. A work published at TCHES 2022 stressed the need for secure higher-order masked comparisons beyond the t-probing model and proposed a higher-order masked comparison method. Subsequently, D'Anvers, Van Beirendonck, and Verbauwhede improved upon the performance of several previous proposals; their higher-order masked algorithm currently achieves the highest performance for masked comparisons.
In this work, we show that while this proposal is secure in the t-probing model, its security in practice is questionable. We first propose an approximate template attack that requires only a small number of traces for profiling and has an exceptionally high noise tolerance. We demonstrate that, without knowledge of the targeted values, a vertical analysis of the distribution of certain points of interest can replace the profiling phase. Finally, we explain how a decryption failure oracle may be constructed from a single trace.
We prove that these attacks are not affected by higher masking orders for noise levels that by far prevent previous profiled attacks on ML-KEM. Further, we provide simulations showing that even under extreme noise levels, the attacks are not prevented by realistic masking orders. Additionally, we carry out the attacks on multiple physical devices to stress the practicality of our attack. We discuss the underlying causes for our attack, demonstrate the difficulty of securing the FO-transform in ML-KEM, draw conclusions about the (in-)sufficiency of t-probing security in this context, and highlight an open gap in securing ML-KEM on embedded devices.

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's security is based on the hardness of lattice problems and has already been adopted by a number of well-known libraries and applications [Jar22, Con23, Ehr23, O'B23]. 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 (FI) and Side Channel Attacks (SCAs), which have been widely researched in the context of ML-KEM [RCDB24]. 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 can 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 sent to the KEM potentially causes a decryption failure that 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 by one bit. An adversary can derive an inequality if they can discriminate between the two cases. 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 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]. This proposal, to the best of our knowledge, currently achieves the best performance and is therefore likely to be used in products.

Our contribution. We show that the proposal of [DBV23] suffers from several practical weaknesses. As in [DHP+22], we do not break any formal 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 1000 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. This allows for template calculation without knowledge of the targeted intermediate values, avoiding a profiling phase. The attack performs only marginally worse than the profiled attack, and also requires about 7000 chosen-ciphertexts and corresponding traces for realistic noise levels.

Third, we demonstrate a horizontal attack that reveals decryption failures by using only a single trace. By analyzing the distributions caused by zero and one bits in the masked ciphertext, it is possible to 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. It allows reducing the security of ML-KEM with just few traces.

We base our attack definition on a theoretical analysis of the distributions of the target routine assuming noisy Hamming weight (HW) leakage. We then explain our attacks using the theoretic model, prove that the required noise levels for masking to mitigate the attacks is very high, give an intuition on what enables these attacks, and discuss the t-probing model in this context. We validate our model with an evaluation on multiple physical devices. To evaluate the proposed approach, we simulate the attacks for multiple noise levels. Our simulations show that the attacks succeed with a noise standard deviation of $\sigma \leq 20$ in the noisy HW model for masking order $t = 4$ and for $\sigma \leq 13$ for $t = 10$.

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. We assume that an adversary has physical access to the target device, knows the public key, is able to submit chosen ciphertexts and can record several thousand traces. Note, that the attack of [DHP+22], motivation for the comparisons of [DHP+22, DBV23], requires a similar number of traces.

The attack can be prevented by a protocol-based countermeasure proposed in [PP21], that locks the device after a certain number of decryption failures. The countermeasure stops all comparable attacks but 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.

The attack of this work 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]. The feasibility of the attack is directly linked to the amount of exploitable side-channel leakage of the multiplication instruction. To the best of our knowledge, there is no open-source ML-KEM implementation utilizing this instruction. We therefore consider the investigation of such an implementation as future work.

Finally, we carry out our non-profiled attacks, assuming that the adversary can select the Point of Interests (PoIs) beforehand. We describe how to select the PoIs using both a labeled and an unlabeled profiling set. We show that PoI selection in this case poses no obstacle for an attacker.

Open source. We publish all resources used for this work[2]. This includes the recorded traces, the leakage simulation, and the implementation of the algorithm for the attacks.

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, a single bit flip in an honestly generated ciphertext can lead to a failure in the decryption process. If the decryption failure can be observed by an adversary, information about the long-term secret key is leaked.

[1] Such an instruction is mentioned in [DBV23] as performance improvement.

[2] Available at `https://github.com/juliusjh/insecurity_masked_comparisons`.

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 CCAs with side-channel or fault analysis to observe decryption failures. We explain the underlying principles of these attacks and provide an overview of the literature. Finally, we reiterate a recent proposal for mitigating SCAs 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 $\mathtt{m}$ $(\mathtt{m}')$. Sampling from a random distribution 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 $\rho$, $\sigma$
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: $\mathbf{r}, \mathbf{e}_1 \in R_q^k \xleftarrow{\$} \text{SampleBinomial}_{\eta_2}(r)$
 4: $e_2 \in R_q \xleftarrow{\$} \text{SampleBinomial}_{\eta_2}(r)$
 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+21b]. Relevant locations for decryption failure attacks are marked in yellow.

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 $k \in \{2, 3, 4\}$. $\mathbf{s}$ is the secret key and the Module Learning with Errors (MLWE) sample $\mathbf{A}$ and $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ form the public key.

Algorithm 4 KEM.KeyGen

```
Require: Randomness seeds $\rho$, $\sigma$
Ensure: Public key $\texttt{pk}$, secret key $\texttt{sk}$
1: $z \xleftarrow{\$} \mathrm{SampleUniform}()$
2: $\texttt{pk}, \texttt{sk}_{\mathrm{pke}} \leftarrow \mathrm{PKE.KeyGen}()$
3: $h = \mathrm{H}(\texttt{pk})$
4: $\texttt{sk} \leftarrow (\texttt{sk}_{\mathrm{pke}}, \texttt{pk}, h, z)$
5: return $\texttt{pk}, \texttt{sk}$
```

Algorithm 5 KEM.Encaps

```
Require: $\texttt{pk}$
Ensure: Ciphertext $\texttt{ct}$, shared secret $\texttt{K}$
1: $\texttt{m} \xleftarrow{\$} \mathrm{SampleUniform}()$
2: $\bar{\texttt{K}}, \texttt{r} \leftarrow \mathrm{G}((\texttt{m}, \mathrm{H}(\texttt{pk})))$
3: $\texttt{ct} \leftarrow \mathrm{PKE.Enc}(\texttt{pk}, \texttt{m}, \texttt{r})$
4: $\texttt{K} \leftarrow \mathrm{KDF}((\bar{\texttt{K}}, \mathrm{H}(\texttt{ct})))$
5: return $\texttt{ct}, \texttt{K}$
```

Algorithm 6 KEM.Decaps

```
Require: $\texttt{sk} = (\texttt{sk}_{\mathrm{pke}}, \texttt{pk}, h, z)$, $\texttt{ct}$
Ensure: Shared secret $\texttt{K}$
1: $\texttt{m}' \leftarrow \mathrm{PKE.Dec}(\texttt{sk}_{\mathrm{pke}}, \texttt{ct}_{\mathrm{pke}})$
2: $\bar{\texttt{K}}', \texttt{r}' \leftarrow \mathrm{G}((\texttt{m}', h))$
3: $\texttt{ct}' \leftarrow \mathrm{PKE.Enc}(\texttt{pk}, \texttt{m}', \texttt{r}')$
4: $b \leftarrow \mathrm{Compare}(\texttt{ct}, \texttt{ct}')$
5: if $b$ then
6:     return $\texttt{K} = \mathrm{KDF}(\texttt{K}', \mathrm{H}(\texttt{ct}))$
7: else
8:     return $\texttt{K} = \mathrm{KDF}(z, \mathrm{H}(\texttt{ct}))$
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+21b]. The comparison operation (red) and the deterministic encryption step (yellow) are highlighted. G and H denote hash functions.

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 coefficients and one bits to $\lceil q/2 \rfloor$ coefficients. The ciphertext consists of the compressed versions of $\mathbf{u}$ and $v + m_{\mathrm{poly}}$. The compression and decompression functions are defined as

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

and

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

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

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

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 $\texttt{m}$ 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 $\texttt{m}$, 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.

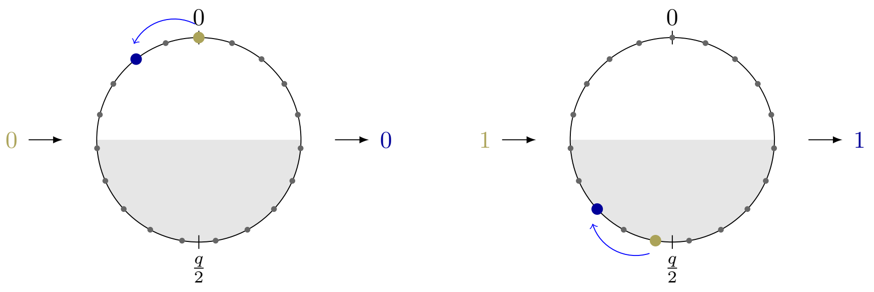
2.1.2 The FO-Transform

Using an FO-transform, a CCA-secure KEM can be constructed from a CPA-secure PKE. While other transformations exist, e.g., [OP01, CHJ+02, CS03, CHK03], ML-KEM [ABD+21b] and all schemes advancing to the fourth round of the NIST PQC standardization [AAC+], as well as the alternatives [ABD+21a, BBC+20] from the third round, make use of the FO-transform (or its variant). In short, the KEM works by encrypting a randomly sampled message using randomness derived from the message itself. During the decapsulation, the other party checks whether the ciphertext was created honestly, by re-encrypting the message and then comparing the resulting ciphertext to the submitted ciphertext.

In more detail: 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 z.

The **encapsulation** first samples a random message $\mathtt{m}$ and encrypts it into a ciphertext $\mathtt{ct}$ using the PKE 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.

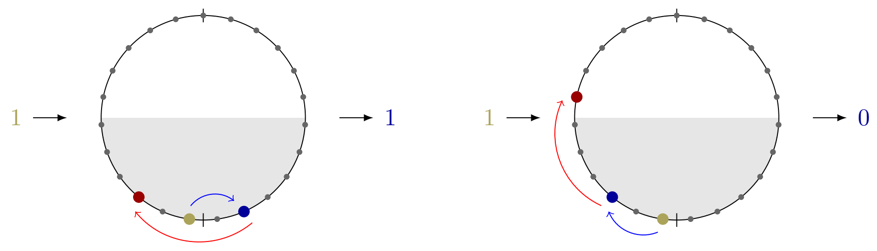
The **decapsulation** starts by invoking the PKE decryption routine to obtain $\mathtt{m}'$ using the secret key. This allows one to obtain the same shared secret as the other party. In addition, the PKE *encryption* routine is called which returns $\mathtt{ct}'$ by encrypting $\mathtt{m}'$. This is also done deterministically with the randomness derived from $\mathtt{m}'$ as in the 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 this setting, the latter prevents observing the former). The KEM routines are shown in Figure 2.



(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].

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 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

Although ML-KEM hides decryption failures, an SCA or a FI may allow adversaries to circumvent this protection (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 may be recovered.

Note that the manipulated ciphertext $\tilde{\texttt{ct}}$ can be chosen to only differ by a single bit from $\texttt{ct}$. However, if a decryption failure occurs, the re-encrypted ciphertext $\texttt{ct}'$ 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 $\texttt{ct}'$ and $\tilde{\texttt{ct}}$ differ in a single bit[5].

2. The re-encrypted ciphertext $\texttt{ct}'$ and $\tilde{\texttt{ct}}$ 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

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

where l is the coefficient in which the error was introduced, and $\texttt{obs}$ 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 as a black box and do not improve upon the method.

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

2.3 The Noisy Hamming Weight Model

As in many previous attacks, our proofs and simulations rely on the noisy HW model: a measurement of a processed value v with HW $h = \mathrm{HW}(v)$ at some point in time leaks as

$$h + \mathcal{N}(0, \sigma), \tag{6}$$

for some σ called the *noise level.* For a 2^n-bit integer, the signal-to-noise ratio (SNR) of a location that leaks h is then

$$\frac{\mathrm{var}(h)}{\sigma^2} = \frac{n}{4\sigma^2}. \tag{7}$$

In Section 5.2.2, we show that our attack succeeds with $\sigma = 20$ in a fourth-order protected implementation in an **unprofiled** setting; this corresponds to a SNR of 0.04 in the noisy HW model. Common profiled attacks (note that two of our attacks are non-profiled) against ML-KEM fail for $\sigma > 2$ or lower [PPM17, PP19, HHP+21], the attack of Kannwischer, Pessl, and Primas [KPP20] against Keccak fails for $\sigma > 3$ and, in their setting, estimates the noise level for an `STM32F405` as "in most usage scenarios [..] well below 3.0" The attack of [HHP+21] is classified as "requiring a reasonably low SNR" by the survey of [RCDB24].

2.4 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.

2.4.1 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.

The authors of [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.

2.4.2 The Masked Comparisons of [DBV23]

In this work, we investigate the side-channel security of the most recent proposal in [DBV23]. By performing the comparison in $\mathbb{F}_2^n$ – basically replacing integer multiplication with Exclusive-Ors (XORs) – the authors improve the performance compared to [DHP+22]; performance benefits over [CGMZ23] are achieved by streamlining a hybrid comparison proposal. Both improvements achieve roughly 20-25% percent improvement alone over the respective previous work, and the authors propose combining the streamlined hybrid method with the Galois Field Comparison algorithm. They 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

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 $\Delta\texttt{bc}$.
Ensure: 0 if the difference is a shared zero, otherwise 1.
1: $E^{(\cdot)} \leftarrow \mathbf{0}$
2: for all $i \in \{0, \ldots, \text{ length } \Delta\texttt{bc} - 1\}$ do
3:     $r \xleftarrow{\$} \{0,1\}^s$
4:     $E^{(\cdot)} \leftarrow E^{(\cdot)} \oplus (r \odot b[i])$
5: end for
6: return $\text{Or}(E^{(\cdot)})$
```

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. In their implementation, s is 64, which provides reasonably low collision probability; we therefore also set $s = 64$ throughout this work.

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 authors propose a bitsliced implementation, which is also how they perform their evaluations. 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 $\Delta\texttt{bc}$[6].

For each shared polynomial coefficient of $\Delta\texttt{bc}$, 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 $\Delta\texttt{bc}$ 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. Note that the parameter s in Algorithm 7 is set to $s = 64$ in the implementation; this will be the case throughout this paper.

In Listing 1, the type of `E` implements a 96-bit integer and has two components: `LSB`, a 64-bit integer, and `MSB`, a 32-bit integer; `biti` is a 32-bit integer holding the currently processed bit of the ciphertext difference, and `tmp` is a 64-bit integer holding $\texttt{biti} \cdot \texttt{R}$. Lines 10 and 11 perform the Galois field addition on a 96-bit integer. The length of $\Delta\texttt{bc}$ is $\texttt{SIMPLECOMPBITS} = 272$; the details on how to obtain $\Delta\texttt{bc}$ from `ct` and $\texttt{ct}'$ are not relevant for our attack, and we therefore refer to [DBV23] for details.

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 the method with respect to performance of the comparison, and prove its security in the t-probing model. We explain how low-effort attacks, which technically do not violate security guarantees, can again target this proposal. 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 sketch the attacks and introduce the adversarial model. Then, we assume a model for distributions at the targeted locations, and compute relevant properties. Based

[6]Implementations often denote $\mathbf{u}$ by `b` and v by `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 $\Delta\mathtt{bc}$ and has length 272 (i.e., SIMPLECOMPBITS=272).

```
 1  for (size_t i = 0; i < SIMPLECOMPBITS; i++)
 2  {
 3      uint64_t R = random_uint64();
 4      for (size_t j = 0; j < NSHARES; j++)
 5      {
 6          for (size_t k = 0; k < 32; k++)
 7          {
 8              biti = (BC_Bitsliced[i][j] >> k) & 1;
 9              tmp = R * biti;
10              E->LSB[j] ^= tmp << k;
11              E->MSB[j] ^= tmp >> (64 - k);
12          }
13      }
14  }
```

on the model, we then propose a template attack [CRR02] that requires a low number of traces during 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 demonstrate that the error introduced by our approximation is rather small. We then show that the profiling phase is not required. The template in our attack is created without knowledge of the targeted intermediate values. We can recover the shares without profiling by estimating the distributions for the targeted values at several locations in the trace. We demonstrate that this horizontal attack only requires a single trace.

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 we base our attacks on. We demonstrate its applicability by capturing side-channel traces of the implementation running on a current microcontroller in Section 4.

3.1 Adversarial Model

We assume that the adversary can submit chosen-ciphertexts and performs a SCA while the decryption is running. The adversary's goal is to relate the chosen-ciphertexts with a decryption failure or success using the traces. This yields inequalities over the secret key, which can be solved by using a biased brute force search method.

3.1.1 Attack sketch.

The adversary creates m chosen-ciphertexts $\tilde{\mathtt{ct}}$ that differ in a single coefficient 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 and cause a decryption failure in statistically half of the cases (and decapsulation failures in all cases). The adversary then records side-channel traces of the "Galois Field Compression" routine (see [DBV23, Algorithm 10], lines 8-11 in Listing 1). The side-channel traces are used to distinguish between decryption failure or success. If the separation rate exceeds 80%, the secret key can be recovered.

Targeted coefficients of $\Delta\mathtt{bc}$. We explain our approach by using a single coefficient of $\Delta\mathtt{bc}$ (see Section 2.4.2). The extension to several coefficients of $\Delta\mathtt{bc}$ is straight-forward.

Required number of traces. The number of traces m has to be chosen based on the attacker's capabilities 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; Table 3 shows that for most noise levels approximately 7000 traces are required.

Differences in adversarial models. All three attacks require the same amount of classified decryption failures or successes. However, the number of traces during the profiling phase differs. If the intermediate values are known during profiling, 500 traces are required. The non-profiled vertical attack requires approximately 1000 traces, but it is not necessary to know the intermediate values. The horizontal attack shows that a single trace is sufficient to build a template by exploiting multiple locations in the trace.

3.1.2 Targeted Routine.

We target the core routine of the comparison proposed by [DBV23] (see Listing 1). The input $\Delta\texttt{bc}$ contains t vectors with 272 32-bit values, where t denotes the number of shares. An adversary aims to recover 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. We aim at recovering the value of `biti` in the inner loop (lines 8-11) of Listing 1.

3.1.3 Leakage Assumption and Notation

In the following, we assume the HW leakage model, and target the inner loop of Listing 1. Let l denote the index of a decapsulation/trace, i denote the index of the current coefficient of $\Delta\texttt{bc}$, j runs over the share indices, and $k \in \{0, \dots, 32\}$ 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 coefficient of $\Delta\texttt{bc}$.

Following the model, we assume that the inner loop of Listing 1 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

$$\texttt{obs}_{l,j,k} \xleftarrow{\$} \mathcal{N}(b_{i,l,j,k} \cdot h_{l,i}, \sigma), \tag{8}$$

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 at a PoI from several traces, ignoring that the $r_{l,i}$ 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 `poi` in a trace l is

$$\texttt{obs}_{l,\texttt{poi}} \xleftarrow{\$} \mathcal{N}(h_{l,i}, \sigma). \tag{9}$$

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

$$\texttt{obs}_{\texttt{poi}} \overset{\$}{\leftarrow} \texttt{Obs}_{l,\texttt{poi}}, \tag{10}$$

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

$$r_{l,i} \overset{\$}{\leftarrow} \{0,1\}^s \tag{11}$$

$$\texttt{obs}_{\texttt{poi}} \overset{\$}{\leftarrow} \mathcal{N}(\mathrm{HW}(r_{l,i}), \sigma). \tag{12}$$

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

$$\sum_{h \in \{0,\dots,s\}} P(h)\mathcal{N}(h,\sigma), \tag{13}$$

where the distribution of the HWs is the sum of RVs 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

$$\texttt{obs}_{l,\texttt{poi}} \overset{\$}{\leftarrow} \mathcal{N}(0,\sigma). \tag{14}$$

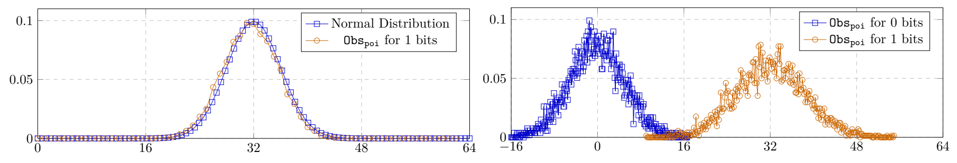
Thus, in this case, $\texttt{Obs}_{\texttt{poi}}$ is normally distributed around 0 with standard deviation σ.

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 number 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.



(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 $\sigma = 5.0$).

If the noise level is sufficiently high and there is a large number of samples, the mixture distribution is very close to a normal distribution because the factors follow a binomial distribution (see Equation (13) 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 $\Delta\mathtt{bc}$ 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 for a target 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.

In the simplest case, the distributions can be separated by using a threshold for the first statistical moment. However, if the distributions overlap, computing the mean and the variance can be done only on the non-overlapping parts of the distributions, this allows for more precise templates. 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 $\Delta\mathtt{bc}$ as well. Note that due to the clear pattern visible in the traces (see Figure 7), a simple brute-force approach is also feasible.

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 the vertical non-profiled attack requires a larger number of trace. 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 $\Delta\mathtt{bc}$. 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 $\Delta\mathtt{bc}$, 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 may then separate the distributions and compute a template without knowledge of $\Delta\mathtt{bc}$. However, in this case, both distribution are Gaussian.

3.4 Recovering Shares and Defeating Masking

In all three attack scenarios, the adversary aims to distinguish between decryption failures and successes given a trace. To achieve this, the adversary has to classify shared bits from observed measurements, obtain the unshared bit, and classify the trace based on the recovered bits.

3.4.1 Classification Algorithm

The simplest algorithm is to just assign the likeliest value to a bit, compute the shared bits, and classify as failure if a certain threshold of 1 bits is crossed – in case of a decryption success, all true values, except the bit flip we introduced, should be 0. However, this may fail due to bit-classification errors, especially in the case of higher-order masking. Instead, we look for an untampered bit that has been classified as 1 with high confidence. If this is the case, a decryption failure is highly probable. Otherwise, we consider all bits that may be classified with a decreasing probability and determine whether the ratio of 0 to 1 bits exceeds or falls below a given threshold for that classification probability. If no bound gets crossed, we decide not to classify the trace at all. Except for the first condition, the algorithm is a heuristic approach. It still performs well during our experiments and is easy to adapt. Adapting the approach to the measured noise levels may enhance classification rates; however, we do not employ such optimizations.

Definition 1. *We call a share of a bit, a bit, or a trace classified if the classification algorithm outputs a value for it; we call it correct if the value matches the true value.*

3.4.2 Effectiveness of Masking.

Masking countermeasures are only proven to be effective if sufficient noise is present (see, e.g., the reduction of [DFS15]). We now show that this masked comparison requires unrealistically high noise for masking to provide a benefit. We here assume the model derived in the previous section and that the adversary can obtain a good approximation for mean and variance for both 0- and 1 bits.

The first reason for our attack's high noise tolerance is that multiplying the targeted bit by a 64-bit integer R increases power consumption and, consequently, leakage. Therefore, the probability of being able to classify a bit b is very high, compared to other SCAs with equal leakage model. For example, as depicted in Figure 5b, for $\sigma = 5.0$, the overlap between the distributions for 0 and 1 bits is fairly small, which means that many bits can be classified with high confidence.

In the following, σ is the standard deviation in the noisy HW model, and measurements for HW h are assumed to be distributed around h (i.e., the mean μ_h is just h); the masking order is denoted by t, and we aim at classifying a shared bit with $p_{\text{share}} > 0.9999$.

Lemma 1. *Let b_j be a share of an unshared bit b of coefficient of $\Delta\mathtt{bc}$ with sampled constant r with $h = \mathrm{HW}(R)$. A classification algorithm that only classifies observations that can be classified with correctness probability $p_{share} > 0.9999$ (4σ interval), can classify more than*

$$p_{class\_sh}(h) = \frac{2}{\sigma\sqrt{2\pi}} \int_{-\infty}^{h-4\sigma} e^{-\frac{x^2}{2\sigma^2}}\, dx \tag{15}$$

of the measurements.

Proof. By model assumption, the distributions for both 0- and 1- bits have the same variance, and the probability density functions intersect in a single point x. The (unprofiled) template attack will classify a 1-bit with measurement `obs` as 0 if and only if `obs` $< x$. To avoid an error, a classification algorithm can choose not to classify measurements for which $\mu_1 - 4\sigma <$ `obs` $< \mu_0 + 4\sigma$. As more than 0.9999 of normally distributed sample lie within the 4σ intervals, the statement follows by symmetry; Figure 6 shows an illustration. □

The second reason for the high amount of noise required for masking to prevent the attack is that the information of whether a decryption failure occurred is encoded in statistically $32 \cdot 271/2$ bits: a single high-confidence classified unshared 1-bit suffices to detect a decryption failure, and for n shares the probability of classifying all shares of a bit is $(1 - p_{\text{share}})^n$, where p_{share} is the probability of recovering a shared bit. This drastically increases the probability of detecting a decryption failure, even if only few bits can be classified because of the high noise.

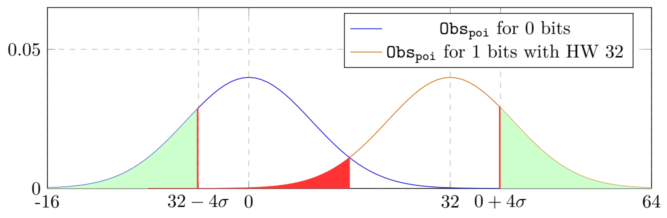


Figure 6: Area (shaded green) in which measurements can be classified with high confidence for HW 32 and $\sigma = 10$ and area (shaded in red) for which 1-bits are classified incorrectly.

Table 1: Lower bounds for the probability of detecting a decryption failure per masking order and standard deviation σ.

shares/σ	≤ 9	10.0	11	12	13	14	15.0	17.0	20
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.39
3	1.00	1.00	1.00	1.00	0.97	0.71	0.37	0.07	0.01
4	1.00	1.00	1.00	0.87	0.40	0.12	0.04	0.00	0.00
8	1.00	0.80	0.14	0.01	0.00	0.00	0.00	0.00	0.00
10	1.00	0.40	0.03	0.00	0.00	0.00	0.00	0.00	0.00
15	1.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	1.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Lemma 2. *The probability of encountering a bit for which all shares can be classified with probability p_{share} is*

$$p_{class\_ush} = 1 - (1 - \sum_{h=0}^{64} \frac{1}{2^{65}} \binom{64}{h} (1 - (1 - \frac{p_{class\_sh}(h)^t}{2})^{32}))^{271} \tag{16}$$

Proof. Fix i, j: For a decryption failure, the probability for $b_{i,j}$ to be 1 is $\frac{1}{2}$, the probability to observe a HW h is $\frac{1}{4}\binom{64}{h}$, the probability that a share of $b_{i,j}$ can be classified is $p_{\text{class\_sh}}(h_i)$. Therefore, the probability of $b_{i,j}$ being 1 and the recovery algorithm being able to classify it correctly with confidence p_{share} is $\sum_{h=0}^{64} \frac{1}{2^{65}} \binom{64}{h} p_{\text{class\_sh}}(h)^t$, which is 0.5 at most. However, there are 32 bits corresponding to the same r_j, thus, the probability of such a bit occurring in a coefficient of $\Delta\texttt{bc}$ is $\sum_{h=0}^{64} \frac{1}{2^{64}} \binom{64}{h} (1 - (1 - \frac{p_{\text{class\_sh}}(h)^t}{2})^{32})$. Now the probability of a bit occurring in any coefficient is given by the term in Equation (16). $\square$

Our main theorem now directly follows:

Theorem 1. *Our attack can classify decryption failures with probability greater than $p_{total} = p_{class\_ush} \cdot p_{share}^t$.*

Values for $p_{\text{class\_ush}}(1 - p_{\text{share}})^t$ are shown in Table 1; (expanded version in Appendix C); note that common **profiled** attacks against **unprotected** implementations are already prevented by noise levels for which masking does not affect our attack.

Note that these are only lower bounds; in our simulations, we also use the algorithm described in Section 3.4.1 with heuristically chosen bounds, which allows us to identify decryption successes as well as in more failures. These bounds also have the advantage of being less affected by non-Gaussian noise and outliers.

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. To demonstrate that these assumptions hold

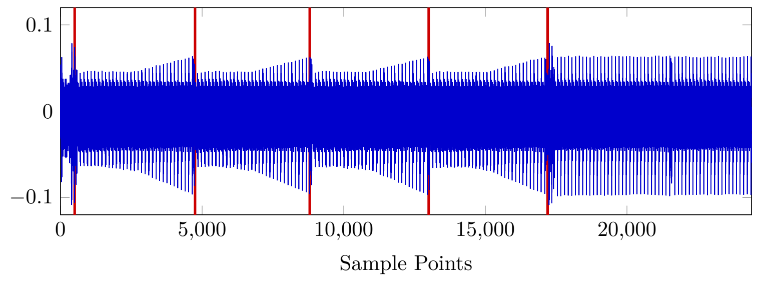


Figure 7: The mean trace (`cw-f4`) showing the first coefficient and parts of the second coefficient of Δ`bc`; approximate locations for the four shares are divided by red vertical lines.

in practice and model a physical device sufficiently well, we execute the proposed attacks on ARM-based microcontrollers.

4.1 Measurement Setup

We used a LeCroy HDO6104A 1 GHz oscilloscope to perform power measurements of a ChipWhisperer CW308 UFO board with an `CW308T-STM32F415` and `CW308T-STM32F3` target [Incb, Incc], referred to as `lecroy-f4` and `lecroy-f3`, respectively. Additional measurements were performed with ChipWhisperer-Lite with an integrated STM32F3 target [Inca], referred to as `cw-f3`, as well as the `CW308T-STM32F415` target measured with an external ChipWhisperer-Lite, referred to as `cw-f4`. All targets ran a custom firmware with the open-source implementation[7] of [DBV23] performing masked comparisons. The firmware was compiled with optimizations enabled, i.e., with the `-O3` compiler flag.

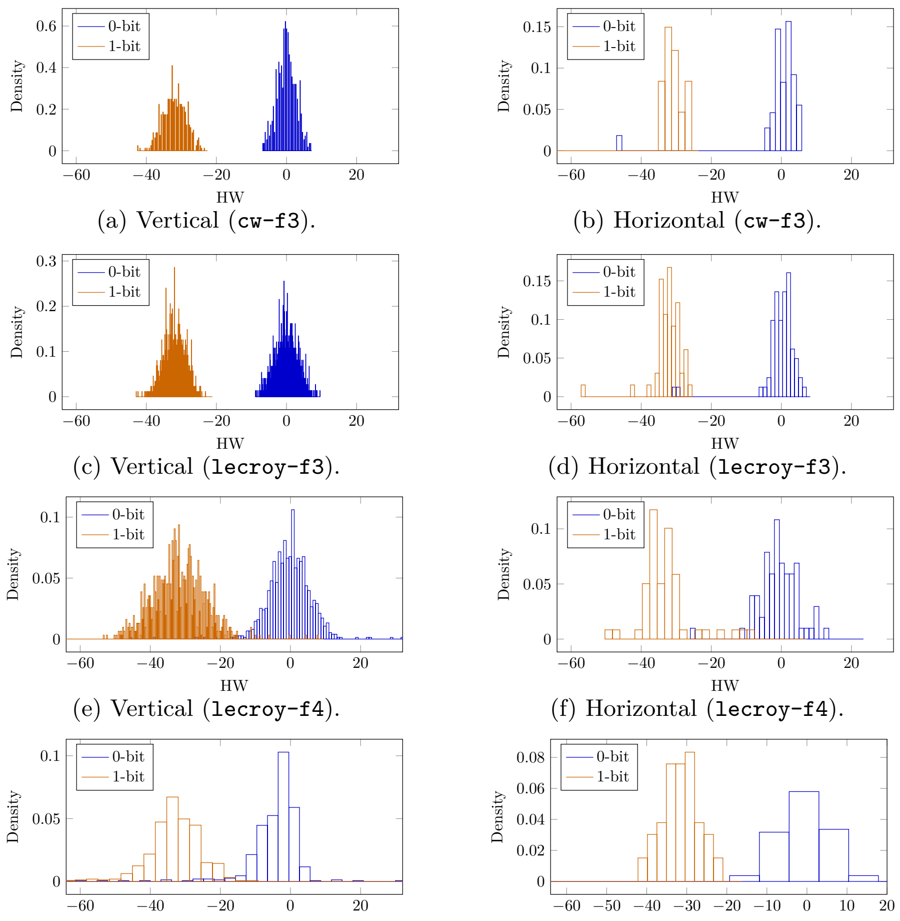
The oscilloscope was connected with an SMA cable, connected to a DC-blocker, filtering out DC and low frequency power supply noise, as well as a chain of one 50 MHz and two 25 MHz low-pass filters. The chain of filters was necessary for the `lecroy-f4` setup, as the STM32F415 based chip of the target incorporates an internal voltage regulator [Incd], which attenuates the power signal and introduces a significant amount of high-frequency noise. To reduce the amount of horizontal noise due to clock jitter, a 10 MHz clock crystal was used to clock the `lecroy-f4` and `lecroy-f3` boards.

The estimated noise levels, given as (scaled) standard deviations, in our attacks are 34.31 (`cw-f4`), 8.33 (`lecroy-f4`), 3.18 (`lecroy-f3`), and 1.89 (`cw-f3`). A real-world adversary could make use of more than one PoI per targeted value and apply various well-established techniques to improve upon the attack, like filtering, trace ordering or other classifiers. However, our straight-forward attack that abstains from any heavy machinery is already sufficient.

4.2 Analyzing the Distributions

We first take a look at the reality of distributions that were modeled in Section 3. 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 Δ`bc`. From the mean of the recorded distributions, inner loops and the individual shares can be clearly identified, as shown in Figure 7. To ensure that we do not exploit any easily detectable implementation mistake, we also performed a t-Test for first order leakage and detected no leakage.

[7] https://github.com/KULeuven-COSIC/Revisiting-Masked-Comparison/.



(a) Vertical (`cw-f3`). (b) Horizontal (`cw-f3`).
(c) Vertical (`lecroy-f3`). (d) Horizontal (`lecroy-f3`).
(e) Vertical (`lecroy-f4`). (f) Horizontal (`lecroy-f4`).
(g) Vertical (`cw-f4`). (h) Horizontal (`cw-f4`).

Figure 8: Vertical and horizontal distributions.

4.2.1 Distributions at 0 and 1 Bits

The vertical and horizontal distribution modeled in Section 3.2 are shown in Figure 8. Here, the vertical distributions are taken over a single PoI for the first targeted bit of the first coefficient of Δ`bc`, and the horizontal distribution is the distribution over all PoI for all bits of the first coefficient of Δ`bc` in the implementation with 4 shares. We scaled the x-axis to match the (negated) HW of the measured value. The distributions for 0- and 1- bits easily identifiable – only a few outliers cannot be clearly attributed to one of the distributions. Some outliers are outside the plotted area, the full plots, including all outliers, can be found in Appendix D. Note that we use merely a single PoI per targeted bit in our attack and do not employ any advanced techniques and heavy machinery known to improve template attacks.

4.3 Carrying Out the Attacks

We evaluate our attack using 1000 (`cw-f3` and `cw-f4`) and 2000 (`lecroy-f3` and `lecroy-f4`) traces. We use 500 traces for profiling in the template attack; the remaining traces are classified into decryption failures and successes. Here, we target only a single coefficient of Δ`bc`; the remaining 270 coefficients may improve the attack. In the attacks we aim at

Table 2: Results in terms of trace classification and correctness rate for our attacks on 3 and 4 shares rounded to two decimal places targeting a single coefficient of $\Delta\texttt{bc}$.

	Attack	Traces (Profil/Attack)	Classified/Correct
`cw-f3`	Template	500/500	1.00/1.00
	Vertical	0/500	1.00/1.00
	Horizontal	0/500	0.99/0.97
`lecroy-f3`	Template	500/1500	1.00/1.00
	Vertical	0/1500	1.00/1.00
	Horizontal	0/1500	0.99/0.96
`lecroy-f4`	Template	500/1500	0.98/0.95
	Vertical	0/1500	0.99/0.95
	Horizontal	0/1500	0.98/0.94
`cw-f4`	Template	500/500	0.92/0.95
	Vertical	0/500	0.96/0.95
	Horizontal	0/500	0.92/0.93

recovering the shares of the bits of $\Delta\texttt{bc}$ as described in Section 3.3 for 4 shares. We then apply the recovery method described in Section 3.4.1 and evaluate the classification and correctness rate of classifying traces into decryption failures and successes. The results in terms of correctness and classification rate (when classifying traces) are show in Table 2.

Adjusting the bounds for decryption failures/successes (c.f., Section 3.4.1) to the observed noise level increases the success rates. We did not employ this optimization and also use the same bounds for the simulations in the next section. In the `lecroy-f4` setup, we disabled the detection of decryption failures using bits that are classified with high confidence; this is because the distributions in this case follow the model less well than in the other setups.

5 Evaluation

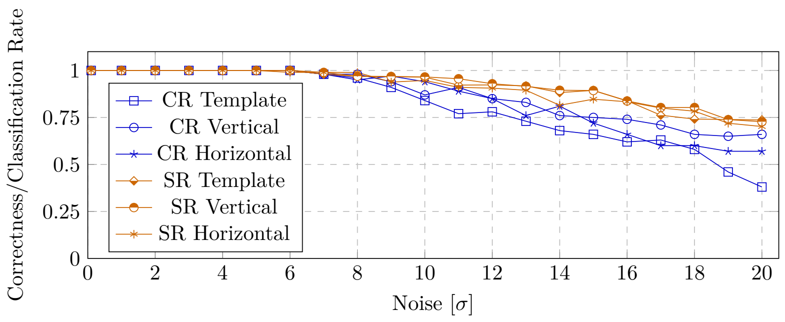
We validated our model by carrying out attacks in practice on physical devices. However, it does only give a rough impression on what noise tolerance the attack exhibits in practice. Thus, in addition to the attacks on a physical device described in the previous section, we evaluate our attacks using simulations using the model described in Section 3.

5.1 Simulation

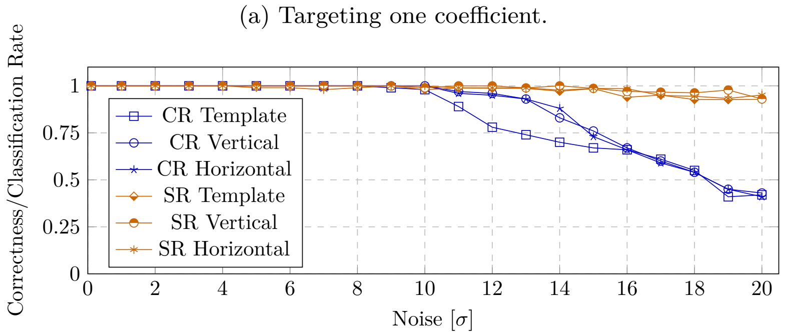
We implemented the model described in Section 3 in Python. We simulate the attack in dependence to the number of coefficients of $\Delta\texttt{bc}$, the number of shares, the number of PoIs per targeted bit, and the number of traces. The noise level is given as standard deviation σ of the used noisy HW model. Given these parameters, we sample simulated traces, by sampling each PoI from the RV defined in Section 3. These simulated traces are used to carry out the same attack as on the physical device's traces.

5.2 Results

In the following, we report on the results using our simulation. We evaluate different settings on 1000 traces for varying noise levels, targeting 1 and 5 coefficients with 4 shares; the correctness and classification rates are obtained by evaluating on the first 100 traces.



(a) Targeting one coefficient.



(b) Targeting five coefficients.

Figure 9: Simulation results in terms of trace classification (CR) and correctness (SR) rate per standard deviation σ.

5.2.1 Distinguishing Decryption Failures and Successes

We first report on the results in terms of distinguishing between decryption failures and successes based on a recorded trace. Note that the full attack does not require us to classify all traces and only requires a correctness rate of more than 0.6[8]. Lower correctness 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.

Results are stated for standard deviation $\sigma \in \{0.1, 1.0, 2.0, \ldots, 20.0\}$. In Figure 9a, we report on the correctness and classification rates for targeting 5 coefficients of $\Delta\texttt{bc}$, and Figure 9b shows the setting in which we target 5 coefficients of $\Delta\texttt{bc}$. For masking order 10, we achieve correctness rate > 0.5 and classifcation rates > 0 – required for a successful attack – up to $\sigma = 13$.

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 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 correctness rate.

Table 3 shows the estimated number of required traces using the implementation of [HMS+23], with ciphertext filtering with `max-delta-v=5`, targeting security level 768 for the obtained classification and correctness rate; the number of required inequalities depends on the correctness rate and the number of required traces depends on the number of required inequalities and the classification rate. The increased classification rate for

[8]The recovery method of [HMS+23] allows for key recovery if more than 0.6 of the inequalities are correct.

Table 3: 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 vertical attack against 4 shares evaluated on 100 traces.

σ	≤ 9	10	15	20
Classification Rate	1.00	1.00	0.76	0.43
Correctness Rate	1.00	0.99	0.99	0.93
Required Inequalities	7000	7000	7000	12000
Required Traces	7000	7000	9500	28000

$\sigma = 20$ compared to $\sigma = 15$ is the result of previously not classified traces being classified incorrectly. Note that in many cases 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[9] of 140. Moreover, even a few correctly classified traces already reduce the security of the instance of the scheme.

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 proposals 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 HWs of the individual shares do not match those of 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

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

shuffled implementation. Further, it is questionable if these additional measures do not eat 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. Whether 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 [PP21]

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 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 $\mathbb{F}_2$. This means that in [DHP+22], all bits of a 32-bit integer leak together, and the adversary now does not learn a single bit but instead the HW of several bits of the ciphertext difference. Therefore, 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). This still gives the adversary information, but leaks much less than processing individual bits. 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 exploited leakage model may not be observable in certain processor architectures if there is an explicit command for Galois field multiplication in the instruction set. In this case, 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, in particular, in the case of the FO comparison.

Acknowledgments

We thank the anonymous reviewers for their insightful comments.

References

[AAC$^+$] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. Status report on the third round of the NIST post-quantum cryptography standardization process.

[ABD$^+$21a] Erdem Alkim, Joppe W. Bos, Leo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, and Douglas Stebila. Frodokem Learning With Errors Key Encapsulation, 2021.

[ABD$^+$21b] 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.

[BBC$^+$20] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. Ntru prime: round 3, 2020.

[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.

[CHJ⁺02] Jean-Sébastien Coron, Helena Handschuh, Marc Joye, Pascal Paillier, David Pointcheval, and Christophe Tymen. GEM: A generic chosen-ciphertext secure encryption method. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, volume 2271 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2002.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[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.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[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.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

[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.

[HHP$^+$21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):88–113, 2021.

[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):287317, 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. Cw1173: Chipwhisperer-lite. https://media.newae.com/datasheets/NAE-CW1173_datasheet.pdf.

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

[Incc] NewAE Technology Inc. Cw308t-stm32f ufo board targets. https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F/.

[Incd] NewAE Technology Inc. Targets with internal regulators. http://wiki.newae.com/Targets_with_Internal_Regulators.

[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.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[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.

[OP01] Tatsuaki Okamoto and David Pointcheval. REACT: rapid enhanced-security asymmetric cryptosystem transform. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175. Springer, 2001.

[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.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

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

[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, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

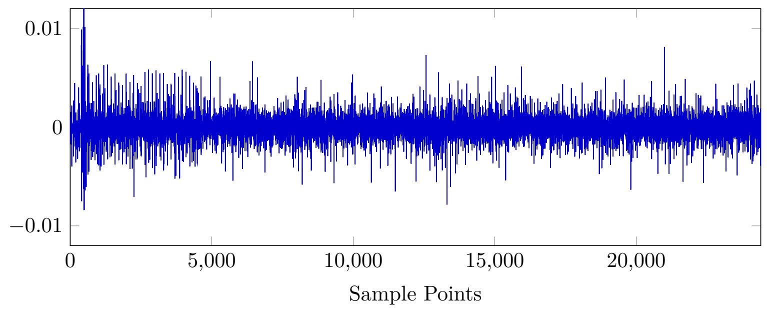
[RCDB24] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter DAnvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. *ACM Trans. Embed. Comput. Syst.*, 23(2), mar 2024.

[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.

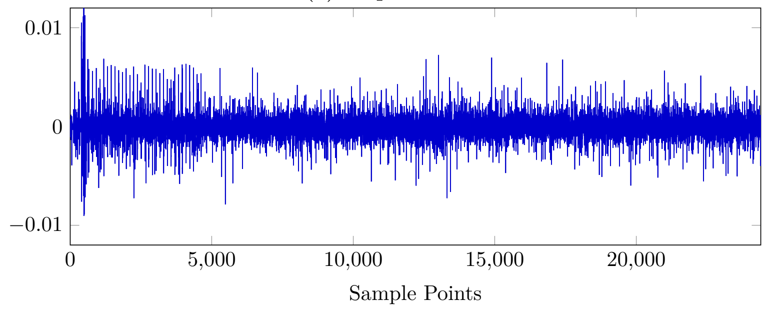
[TH08] Stefan Tillich and Christoph Herbst. Attacking state-of-the-art software countermeasures-a case study for AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, 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.



(a) Unprofiled



(b) Profiled

Figure 10: 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 Δ`bc`. Both plots were obtained in the setting of Section 4 with 3 shares (`cw-f4`).

A 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 to the model we presented in this section. Under this assumption, they may separate the traces into two classes, and abort if one class contains significantly more samples than the other. If both sets contain a similar number 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 10, 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 10 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.

B 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. The results are presented in Figure 11 and show no significant first-order leakage. The red line marks a t-score value of 4.5, which is a common bound

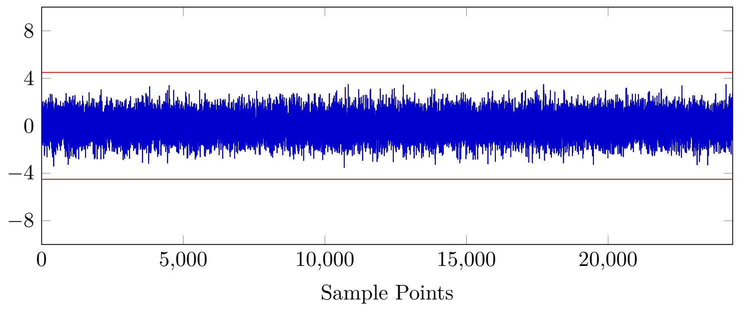


Figure 11: T-test statistics (`cw-f4`); bound of 4.5 in red.

Table 4: Lower bounds for probabilities of taking a measurement of a shared bit that can be classified with confidence > 0.9999 when the Hamming weight is 32 in a single coefficient, of an unshared bit in any coefficient that can be classified with confidence > 0.9999, and of being able to detect a decryption failure.

shares/σ	5.0	6.0	7.0	8.0	10.0	15.0	17.0	20
1	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/1.00/1.00	0.02/1.00/1.00	0.01/1.00/1.00
2	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/1.00/1.00	0.02/0.89/0.89	0.01/0.39/0.39
3	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/0.37/0.37	0.02/0.07/0.07	0.01/0.01/0.01
4	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/0.04/0.04	0.02/0.00/0.00	0.01/0.00/0.00
5	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
6	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/1.00/1.00	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
8	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.80/0.80	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
10	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.40/0.40	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
12	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.17/0.17	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
15	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.05/0.05	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
17	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.02/0.02	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00
20	0.99/1.00/1.00	0.91/1.00/1.00	0.72/1.00/1.00	0.50/1.00/1.00	0.21/0.01/0.01	0.03/0.00/0.00	0.02/0.00/0.00	0.01/0.00/0.00

to assume that no leakage is present. 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 $\Delta\texttt{bc}$.

C Recovery Probabilities

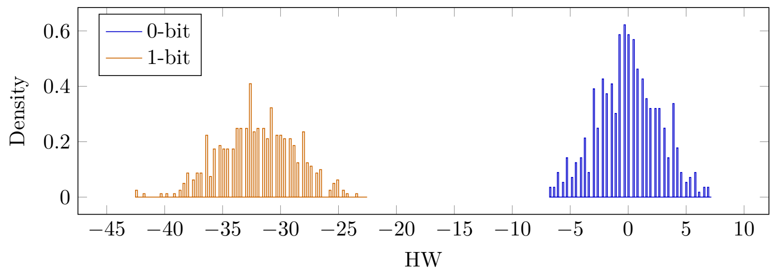
Table 4 shows the probabilities $p_{\text{class\_sh}}(32)$, $p_{\text{class\_ush}}$, and $p_{\text{total}} = p_{\text{class\_sh}}(1 - p_{\text{share}})^n$ from Section 3.4.2. The entries per sigma and shares have the format $p_{\text{class\_sh}}(32)/p_{\text{class\_ush}}/p_{\text{total}}$

D Plots of Distribution

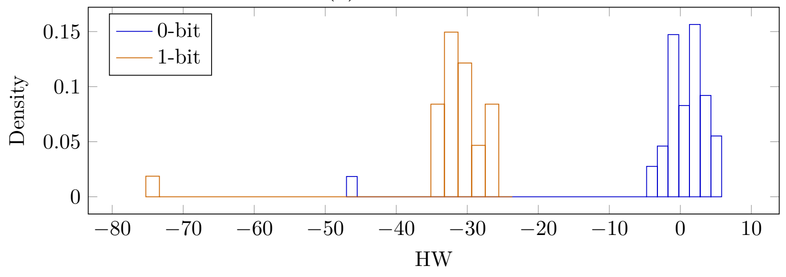
The plots in this section show the full versions of the plots in Section 4.2.1. Figure 12 shows the plots for the `cw-f3` setup, Figure 13 for the `lecroy-f3` setup, Figure 14 for the `lecroy-f4` setup, and Figure 15 for the `cw-f4` setup.

E Results for $t = 10$

Figure 16 shows the simulations results in terms of classification an correctness rate when targeting 5 coefficients of $\Delta\texttt{bc}$ for a $t = 10$ order masked implementation.

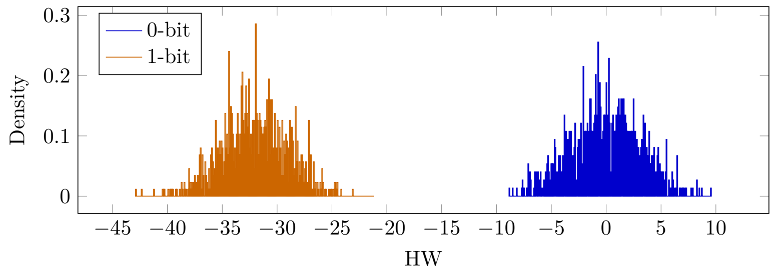


(a) Vertical.

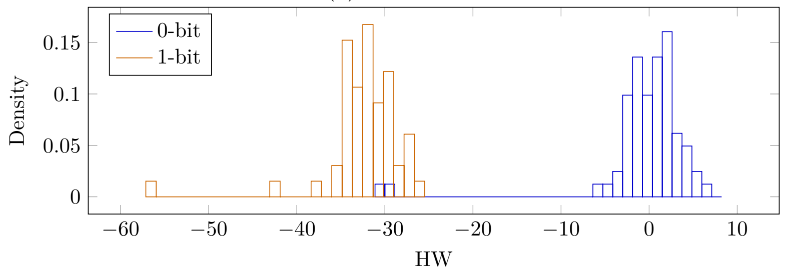


(b) Horizontal.

Figure 12: Vertical and horizontal distributions (`cw-f3`).

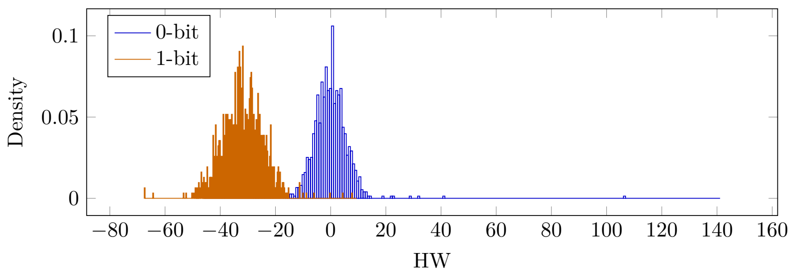


(a) Vertical.

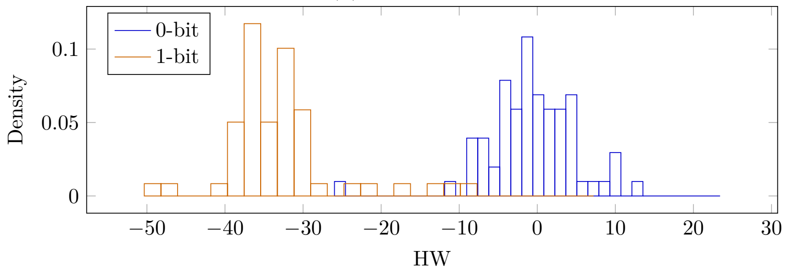


(b) Horizontal.

Figure 13: Vertical and horizontal distributions (`lecroy-f3`).

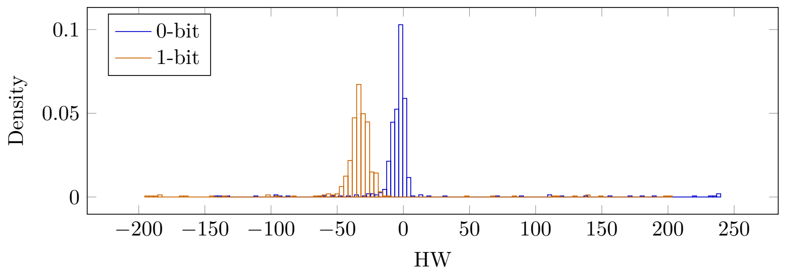


(a) Vertical.

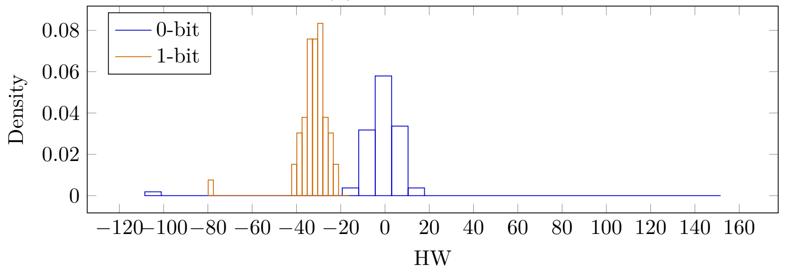


(b) Horizontal.

Figure 14: Vertical and horizontal distributions (`lecroy-f4`).



(a) Vertical.



(b) Horizontal.

Figure 15: Vertical and horizontal distributions (`cw-f4`).

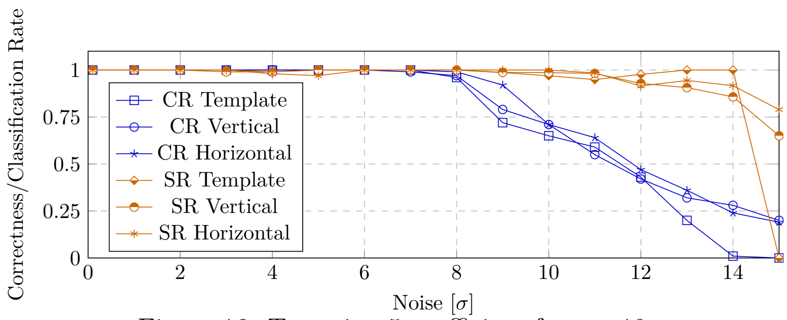


Figure 16: Targeting 5 coefficient for $t = 10$.