Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results

PRASANNA RAVI* and ANUPAM CHATTOPADHYAY†, Temasek Labs, Nanyang Technological University, Singapore and School of Computer Science and Engineering, Nanyang Technological University, Singapore
ANUBHAB BAKSI‡, Temasek Labs, Nanyang Technological University, Singapore

In this work, we present a systematic study of Side-Channel Attacks (SCA) and Fault Injection Attacks (FIA) on structured lattice-based schemes, with main focus on Kyber Key Encapsulation Mechanism (KEM) and Dilithium signature scheme, which are leading candidates in the NIST standardization process for Post-Quantum Cryptography (PQC). Through our study, we attempt to understand the underlying similarities and differences between the existing attacks, while classify them into different categories. Given the wide-variety of reported attacks, simultaneous protection against all the attacks requires to implement customized protections/countermeasures for both Kyber and Dilithium. We therefore present a range of customized countermeasures, capable of providing defenses/mitigations against existing SCA/FIA. Amongst the presented countermeasures, we propose two *novel* countermeasures to protect Kyber KEM against SCA and FIA assisted chosen-ciphertext attacks. We implement the presented countermeasures within two well-known public software libraries for PQC - (1) *pqm4* library for the ARM Cortex-M4 based microcontroller and (2) *liboqs* library for the Raspberry Pi 3 Model B Plus based on the ARM Cortex-A53 processor. Our performance evaluation reveals that the presented custom countermeasures incur reasonable performance overheads, on both the evaluated embedded platforms. We therefore believe our work argues for usage of custom countermeasures within real-world implementations of lattice-based schemes, either in a standalone manner, or as reinforcements to generic countermeasures such as masking.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures**.

Additional Key Words and Phrases: Lattice-based Cryptography, Side-Channel Attacks, Fault-Injection Attacks, Kyber, Dilithium

1 INTRODUCTION

The NIST standardization process for post-quantum cryptography is currently in the third and final round with seven finalist candidates and eight alternate candidates for Public Key Encryption (PKE), Key Encapsulation Mechanisms (KEM) and Digital Signatures (DS) [1]. Theoretical post-quantum security guarantees and implementation performance on different HW/SW platforms served as the primary criteria for selection in the initial rounds of the NIST standardization process. However, resistance against side-channel attacks (SCA) and fault injection attacks (FIA) as well as the cost of implementing protections against SCA and FIA, has also emerged as a very important criterion towards the latter part of the standardization process. This is especially true, when it comes to comparing schemes with tightly matched security and efficiency [4]. In [1, Sections 3.4 and 2.2.3] NIST states that it *encourages additional research regarding side-channel analysis* of the finalist candidates and *hopes to collect more information about the costs of implementing these algorithms in a way that provides resistance to such attacks.*

Three out of the seven finalist candidates derive their hardness from the well-known Learning With Error (LWE) and Learning With Rounding (LWR) problem that operate structured lattices. In particular, SCA and FIA of schemes such as Kyber [5], Saber [17] and Dilithium [20] has received considerable attention with several works demonstrating practical attacks [11, 39, 40, 49], particularly on embedded targets. These attacks have been realized using a wide-range of attack vectors such as power and Electromagnetic Emanation (EM) for SCA and voltage/clock glitching and EM for

Authors' addresses: Prasanna Ravi, prasanna.ravi@ntu.edu.sg; Anupam Chattopadhyay, anupam@ntu.edu.sg, Temasek Labs, Nanyang Technological University, Singapore and School of Computer Science and Engineering, Nanyang Technological University, Singapore; Anubhab Baksi, anubhab.baksi@ntu.edu.sg, anubhab.baksi@ntu.edu.sg, Temasek Labs, Nanyang Technological University, Singapore.

FIA. The proposed attacks have been quite diverse in nature, in terms of the targeted operation, method of constructing queries to the target device, as well as the mathematical approach for key recovery.

There are existing works such as [34, 44] that have provided a good overview of existing implementations of PQC. However, a similar overview that covers recent developments in the field of SCA and FIA of PQC, and in particular, lattice-based schemes is missing. This is especially important, given the ever-growing list of attacks proposed on practical embedded implementations of lattice-based schemes. As a *first contribution*, we present a systematic study of SCA/FIA mounted on lattice-based schemes, with main focus on two leading candidates based on variants of the LWE problem - Kyber KEM [5] and Dilithium signature scheme [20]. Through our study, we attempt to understand the underlying similarities and differences between the existing attacks, while classify them into different categories.

On a parallel front, there has also been significant interest in the cryptogaphic community towards development of SCA and FIA countermeasures for lattice-based schemes. They can be broadly classified into two categories - (1) Generic and (2) Custom. Generic countermeasures attempt to provide concrete security guarantees agnostic to the attack strategy, while custom countermeasues are those that offer protection against specific targeted attacks. With respect to SCA, there have been several works that have proposed generic masking stratgies for lattice-based schemes [10, 14, 37]. However, one can observe several shortcomings with respect to adopting generic countermeasures such as masking. Firstly, practical attacks have been demonstrated over masked implementations of lattice-based schemes [35], and non-trivial flaws in theoretically secure masking schemes have also been exploited for key-recovery [11]. Secondly, masking has been shown to result in significant performance overheads for both lattice-based KEMs as well as signature schemes, especially on embedded software platforms [10, 14].

This leads us to question, if generic countermeasures such as masking alone are sufficient to offer concrete protection against SCA, with minimal overhead. If not, what are the specific countermeasures that can be incorporated to provide additional protection to harden concrete masking schemes. The same question can also be posed with respect to FIA over lattice-based schemes. In this respect, as a *second contribution*, we present a range of customized countermeasures for Kyber and Dilithium, in an attempt to provide protection against most of the SCA and FIA reported over these two schemes. Notably, we also propose two *novel* countermeasures, namely CT_Sanity_Check and Message_Poly_Sanity_Check, for the decryption procedure of Kyber KEM, that offers simultaneous protection against SCA and FIA assisted chosen-ciphertext attacks, which form a major category of the reported SCA and FIA on Kyber KEM.

Finally, we also implement all the presented countermeasures in this work, within two well-known public software libraries for PQC - (1) *pqm4* library for the ARM Cortex-M4 based microcontroller [29] and (2) *liboqs* library [52] for the Raspberry Pi 3 Model B Plus based on the ARM Cortex-A53 processor. Our performance evaluation reveals that the presented custom countermeasures incur reasonable performance overheads, on both the evaluated embedded platforms. We therefore believe our work argues for usage of custom countermeasures within real-world implementations of lattice-based schemes, either in a standalone manner, or as reinforcements to generic countermeasures such as masking.

2 BACKGROUND

2.1 Notations

Elements in the integer ring $\mathbb{Z}_q$ are denoted by regular font letters viz. $a, b \in \mathbb{Z}_q$, where q is a prime. The i^{th} bit in an element $x \in \mathbb{Z}_q$ is denoted as x_i. Vectors and matrices of integers in $\mathbb{Z}_q$ (i.e.) $\mathbb{Z}_q^k$ and $\mathbb{Z}_q^{k \times \ell}$ are denoted in bold upper case letters. The polynomial ring $\mathbb{Z}_q(x)/\phi(x)$ is denoted as R_q where $\phi(x) = (x^n + 1)$ is its reduction polynomial. We denote

$\mathbf{r} \in R_q^{k \times \ell}$ as a *module* of dimension $k \times \ell$. Polynomials in R_q and vectors of polynomials in R_q^k are denoted in bold lower case letters. Matrices of integers of polynomials (i.e.) $R_q^{k \times \ell}$ are denoted in bold upper case letters. The i^{th} coefficient of a polynomial $\mathbf{a} \in R_q$ is denoted as $\mathbf{a}[i]$ and the i^{th} polynomial of a given module $\mathbf{x} \in R_q^k$ as $\mathbf{x}_i$. Multiplication of two polynomials $\mathbf{a}$ and $\mathbf{b}$ in the ring R_q is denoted as $\mathbf{c} = \mathbf{a} \cdot \mathbf{b} \in R_q$ or $\mathbf{a} \times \mathbf{b} \in R_q$. Byte arrays of length n are denoted as $\mathcal{B}^n$. Pointwise/Coefficient-wise multiplication of two polynomials $(a, b) \in R_q$ is denoted as $c = a \circ b \in R_q$. For a given element a ($\mathbb{Z}_q$ or R_q or $R_q^{k \times \ell}$), its corresponding faulty value is denoted as a^* and we utilize this notation throughout the paper. The NTT representation of a polynomial $a \in R_q$ is denoted as $\hat{a} \in R_q$, and the same notation also applies to modules of higher dimension.

2.2 The Learning With Errors Problem [50]

The hardness of both Kyber and Dilithium are based on variants of the well-known Learning With Errors (LWE) problem. The central component of the LWE problem is the LWE instance.

Definition 2.1 (LWE Instance). *For a given dimension $n \geq 1$, elements in $\mathbb{Z}_q$ with $q > 2$ and a Gaussian error distribution $\mathcal{D}_\sigma(\cdot)$, an LWE instance is defined as the ordered pair $(\mathbf{A}, T) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where $\mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ and $T = \mathbf{A} \cdot \mathbf{S} + E$ with $\mathbf{S} \leftarrow \mathcal{D}_\sigma(\mathbb{Z}_q^n)$ and $E \leftarrow \mathcal{D}_\sigma(\mathbb{Z}_q)$.*

Given an LWE instance, one can define two variants of the LWE problem - (1) *Search LWE* problem - Given polynomially many LWE instances $(\mathbf{A}, T) \in (\mathbf{Z}_q^n, \mathbf{Z}_q)$, solve for $\mathbf{S} \in \mathbb{Z}_q^n$ and (2) *Decisional LWE* - Given many random instances belonging to either (1) valid LWE instances $(\mathbf{A}, T) \in (\mathbf{Z}_q^n, \mathbf{Z}_q)$ or (2) uniformly random instances drawn from $\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$, distinguish the valid LWE instances from randomly selected ones.

Cryptographic schemes built upon the standard LWE problem suffered from quadratic key sizes and computational times in the dimension n of the lattice (i.e.) $\mathcal{O}(n^2)$ [50]. Thus, most of the lattice-based schemes, especially those in the NIST standardization process are based on *algebraically structured* variants of the standard LWE and LWR problem known as the Ring/Module-LWE (RLWE/MLWE) problems respectively. The ring variant of the LWE problem (RLWE) [31] deals with computation over polynomials in polynomial rings $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ with $\mathbf{s}, \mathbf{e} \leftarrow \mathcal{D}_\sigma(R_q)$ such that the corresponding RLWE instance is defined as $(\mathbf{a}, \mathbf{t} = \mathbf{a} \times \mathbf{s} + \mathbf{e}) \in (R_q \times R_q)$. The module variant deals with computations over vectors/matrices of polynomials in $R_q^{k_1 \times k_2}$ with $(k_1, k_2) > 1$. With $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k_1 \times k_2})$ and $\mathbf{s} \leftarrow \mathcal{D}_\sigma(R_q^{k_2})$ and $\mathbf{e} \leftarrow \mathcal{D}_\sigma(R_q^{k_1})$. the corresponding MLWE instance is defined as $(\mathbf{a}, \mathbf{t} = \mathbf{a} \times \mathbf{s} + \mathbf{e}) \in (R_q^{k_1 \times k_2}, R_q^{k_2})$.

2.3 Number Theoretic Transform (NTT) based Polynomial Multiplication

Polynomial multiplication is one of the most computationally intensive operations in structured lattice-based schemes such as Kyber and Dilithium. Both Kyber and Dilithium are designed with parameters that allow the use of the well-known Number Theoretic Transform (NTT) for polynomial multiplication. The NTT is simply a bijective mapping for a polynomial $p \in R_q$ from a *normal* domain into an alternative represetation $\hat{p} \in R_q$ in the *NTT domain* as follows:

$$\hat{p}_j = \sum_{i=0}^{n-1} p_i \cdot \omega^{i \cdot j} \tag{1}$$

where $j \in [0, n-1]$ and ω is the n^{th} root of unity in the operating ring $\mathbb{Z}_q$. The corresponding inverse operation named Inverse NTT (denoted as INTT) maps $\hat{p}$ in the NTT domain back to p in the normal domain. The use of NTT requires either the n^{th} root of unity (ω) or $2n^{\text{th}}$ root of unity (ψ) in the underlying ring $\mathbb{Z}_q$ ($\psi^2 = \omega$), which can be ensured through appropriate choices for the parameters (n, q). The powers of ω and ψ that are used within the NTT

computation are commonly referred to as *twiddle constants*. NTT based multiplication of two polynomials $\mathbf{a}$ and $\mathbf{b}$ in R_q is typically done as follows:

$$\mathbf{c} = \text{INTT}(\text{NTT}(\mathbf{a}) \circ \text{NTT}(\mathbf{b})). \tag{2}$$

The NTT over an n point sequence is performed using the well-known *butterfly* network, which operates over $\log_2(n)$ stages. Refer to the algorithmic specification document of Kyber and Dilithium, on more information about the NTT used in the respective schemes [5, 20].

2.4 Kyber

2.4.1 Algorithmic Description. Kyber is a chosen-ciphertext secure (CCA-secure) KEM based on the Module-LWE problem and is considered to be a promising candidate owing to its strong theoretical security guarantees and implementation performance [5]. Computations are performed over modules in dimension $(k \times k)$ (i.e) $R_q^{k \times k}$. Kyber provides three security levels with `Kyber-512` (NIST Security Level 1), `Kyber-768` (Level 3) and `Kyber-1024` (Level 5) with $k = 2, 3$ and 4 respectively. Kyber operates over the anti-cyclic ring R_q with a prime modulus $q = 3329$ and degree $n = 256$, which allow the use of Number Theoretic Transform (NTT) for polynomial multiplication. The CCA-secure Kyber contains in its core, a chosen-plaintext secure Kyber encryption scheme `Kyber.CPA` which is based on the well-known framework of the LPR encryption scheme [31].

Refer to Algorithm 1 for a simplified description of the key-generation, encryption and decryption procedures of CPA secure PKE of Kyber. The function Sample_U samples from a uniform distribution, Sample_B samples from a binomial distribution; Expand expands a small seed into a uniformly random matrix in $R_q^{k \times k}$. The function $\text{Compress}(u, d)$ lossily compresses $u \in \mathbb{Z}_q$ into $v \in \mathbb{Z}_{2^d}$ with $q > 2^d$, while $\text{Decompress}(v, d)$ extrapolates $v \in \mathbb{Z}_{2^d}$ into $u' \in \mathbb{Z}_q$.

Security and Correctness of CPA Secure Kyber Encryption Scheme

The key generation procedure of `Kyber.CPA` PKE simply involves generation of an LWE instance $(\mathbf{A}, \mathbf{t}) \in (R_q^{k \times k} \times R_q^k)$ where $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ (Line 9 in Alg.1). The module $\mathbf{A}$ is sampled from a uniform distribution, while the secrets and errors $\mathbf{s}, \mathbf{e}$ are sampled from a CBD distribution. Given that NTT is used for polynomial multiplication, the public key and secret key are directly represented in the NTT domain. The LWE instance $(\mathbf{A}, \mathbf{t})$ is the public key, while the secret $\mathbf{s}$ forms the secret key.

The encryption procedure involves generation of two LWE instances $\mathbf{u}, \mathbf{v} \in (R_q^k \times R_q)$. The first LWE instance is generated as $\mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e_1}$ (Line 18) and the second LWE instance is generated as $\mathbf{v}_p = \mathbf{t}^T \cdot \mathbf{r} + \mathbf{e}_2$ (Line 19). Subsequently, the message to be encrypted (i.e.) $m \in \mathcal{B}^*$ is encoded into a message polynomial $\mathbf{m} \in R_q$, one bit at a time, in the following manner. If a message bit $m_i = 1$, then the corresponding coefficient $\mathbf{m}[i] = \lceil q/2 \rfloor$, else $\mathbf{m}[i] = 0$ otherwise. Then, this message polynomial is additively hidden within $\mathbf{v}_p$ as $\mathbf{v} = \mathbf{v}_p + \mathbf{m}$ (Line 20). Subsequently, the coefficients of $\mathbf{u}$ and $\mathbf{v}$ are lossily compressed to varying degrees (i.e.) d_1 and d_2 bits respectively, and the compressed versions of $\mathbf{u}, \mathbf{v}$ form the ciphertext ct (Line 21).

The decryption procedure extracts the polynomials $\mathbf{u}'$ and $\mathbf{v}'$ from the ciphertext ct with $\Delta\mathbf{u} = \mathbf{u}' - \mathbf{u}$ (resp. $\mathbf{v}$). Subsequently, the decryption procedure computes $\mathbf{m}' = \mathbf{v}' - \mathbf{u}' \cdot \mathbf{s}$ (Line 26), which is nothing but an approximation of

Algorithm 1: CPA Secure Kyber PKE (Simplified)

procedure CPA.KeyGen
  $seed_A \in \mathcal{B} \leftarrow \mathsf{Sample}_U()$ ▷ Generate uniform $Seed_A$
  $seed_B \in \mathcal{B} \leftarrow \mathsf{Sample}_U()$ ▷ Generate uniform $Seed_B$
  $\hat{\mathbf{A}} = \mathsf{NTT}(\mathbf{A}) \in R_q^{k \times k} \leftarrow \mathsf{Expand}(seed_A)$ ▷ Expand $seed_A$ into $\hat{\mathbf{A}}$ in NTT domain
  $\mathbf{s} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_B, coins_s)$ ▷ Sample secret $\mathbf{s}$ using $(Seed_B, coins_s)$
  $\mathbf{e} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_B, coins_e)$ ▷ Sample error $\mathbf{e}$ using $(Seed_B, coins_e)$
  $\hat{\mathbf{s}} \in R_q^k \leftarrow \mathsf{NTT}(\mathbf{s})$ ▷ $\mathsf{NTT}(\mathbf{s})$
  $\hat{\mathbf{e}} \in R_q^k \leftarrow \mathsf{NTT}(\mathbf{e})$ ▷ $\mathsf{NTT}(\mathbf{e})$
  $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$ ▷ $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ in NTT domain
  Return $(pk = (seed_A, \hat{\mathbf{t}}), sk = (\hat{\mathbf{s}}))$
end procedure

procedure CPA.Encrypt$(pk, m \in \{0,1\}^{256}, seed_R \in \{0,1\}^{256})$
  $\hat{\mathbf{A}} \in R_q^{k \times k} \leftarrow \mathsf{Expand}(seed_A)$
  $\mathbf{r} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_R, coins_0)$ ▷ Sample $\mathbf{r}$ using $(Seed_R, coins_0)$
  $\mathbf{e_1} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_R, coins_1)$ ▷ Sample $\mathbf{e}_1$ using $(Seed_R, coins_1)$
  $\mathbf{e_2} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_R, coins_2)$ ▷ Sample $\mathbf{e}_2$ using $(Seed_R, coins_2)$
  $\hat{\mathbf{r}} \in R_q^k \leftarrow \mathsf{NTT}(\mathbf{r})$ ▷ $\mathsf{NTT}(\mathbf{r})$
  $\mathbf{u} \in R_q^k \leftarrow \mathsf{INTT}(\mathbf{A}^T \circ \hat{\mathbf{r}}) + \mathbf{e_1}$ ▷ $\mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1$
  $\mathbf{v}_p \in R_q \leftarrow \mathsf{INTT}(\hat{t}^T \circ \hat{\mathbf{r}}) + \mathbf{e_2}$ ▷ $\mathbf{v}_p = \mathbf{t}^T \cdot \mathbf{r} + \mathbf{e}_2$
  $\mathbf{v} = \mathbf{v}_p + \mathsf{Encode}(m)$
  Return $ct = \mathsf{Compress}(\mathbf{u}, d_1), \mathsf{Compress}(\mathbf{v}, d_2)$
end procedure

procedure CPA.Decrypt(sk, ct)
  $\mathbf{u}' \in R_q^k = \mathsf{Decompress}(\mathbf{u}, d_1)$; $\mathbf{v}' \in R_q^k = \mathsf{Decompress}(\mathbf{v}, d_2)$
  $\hat{\mathbf{u}}' = \mathsf{NTT}(\mathbf{u}')$
  $\mathbf{m}' \in R_q = \mathbf{v}' - \mathsf{INTT}(\hat{\mathbf{u}}' \circ \hat{\mathbf{s}})$ ▷ $\mathbf{m}' = \mathbf{v}' - \mathbf{u}' \cdot \mathbf{s}$
  $m' \in \mathcal{B}^* = \mathsf{Decode}(\mathbf{m}')$
  Return m'
end procedure

the message polynomial $\mathbf{m}$ (i.e.) $\mathbf{m}'$. This is given as follows:

$$\begin{aligned}
\mathbf{m}' &= \mathbf{v}' - \mathbf{s}^T \cdot \mathbf{u}' \\
&= \mathbf{v} + \Delta\mathbf{v} - (\mathbf{u} + \Delta\mathbf{u}) \cdot \mathbf{s} \\
&= \mathbf{t}^T \cdot \mathbf{r} + \mathbf{e}_2 + \Delta\mathbf{v} + \mathsf{Encode}(m) - \mathbf{s}^T \cdot (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1 + \Delta\mathbf{u}) \\
&= \mathsf{Encode}(m) + (\mathbf{e}^T \cdot \mathbf{r} + \mathbf{e}_2 + \mathbf{s}^T \cdot \mathbf{e}_1 + \mathbf{s}^T \cdot \Delta\mathbf{u} + \Delta\mathbf{v}) \\
&= \mathsf{Encode}(m) + \mathbf{d}
\end{aligned} \tag{3}$$

where $\mathbf{d} = (\mathbf{e}^T \cdot \mathbf{r} + \mathbf{e}_2 + \mathbf{s}^T \cdot \mathbf{e}_1^T + \mathbf{s}^T \cdot \Delta\mathbf{u} + \Delta\mathbf{v})$ is the noise component in $\mathbf{m}'$, which is also linearly dependent on the secret and error $(\mathbf{s}, \mathbf{e})$ of the public-private key pair. The approximate message polynomial $\mathbf{m}'$ is decoded into the message $m' \in \mathcal{B}^*$ one bit at a time in the following manner: If a given message coefficient $\mathbf{m}[i]$ is in the range

$[q/4, 3q/4]$, then $m_i = 1$, else $m_i = 0$ otherwise (Line 27). This is computed using a specialized decoding routine, which is sketched in the code snippet shown in Fig.1. It takes as input the message polynomial $\mathbf{m}$ and decodes the coefficients, one at a time into corresponding bits in the 32-byte message array m.

```
1 uint16_t t = (((m->coeffs[8*i+j] << 1) + KYBER_Q/2) / KYBER_Q) & 1;
2 m[i] |= t << j;
```

Fig. 1. Message Decoding Routine in Kyber KEM, which converts the message polynomial $\mathbf{m} \in R_q$ into a 32-byte message array m, where i denotes the byte location and j denotes the bit location within a given byte.

As long as the absolute value of all the coefficients of the noise $\mathbf{d}$ are less than $q/4$ (i.e.) $\ell_\infty(\mathbf{d}) < q/4$, the message polynomial $\mathbf{m'}$ is decoded to the correct message m (i.e.) $m' = m$. The parameters of the scheme are chosen so as to attain a negligible decryption failure probability. For recommended parameters of Kyber, the decryption failure probability is $\approx 2^{-164}$. While we have only presented a simplified description of Kyber PKE, we refer the reader to [5] for a detailed description of the same.

2.4.2 CCA Transformation. The aforementioned PKE is only secure against chosen-plaintext attacks (CPA secure), and thus can be broken in a chosen-ciphertext setting. The CPA secure Kyber is converted into a CCA-secure Kyber KEM using the well-known Fujisaki-Okamoto transformation [23]. It utilizes a pair of hash functions $\mathcal{H}$ and $\mathcal{G}$ and forms a wrapper around the encryption and decryption procedures, resulting in encapsulation and decapsulation procedures of a CCA secure KEM (Refer Alg.2).

In theory, the FO transform helps protect the decapsulation procedure of KEMs against chosen-ciphertext attacks in the following manner. The message m' obtained after decryption of the received ciphertext ct (Line 17) is hashed with the public key to generate a pre-shared secret $\bar{K}'$ and a seed r (Line 18). The message m' along with the seed r is fed into a re-encryption procedure to recompute the ciphertext as ct' (Line 20). A subsequent comparison of ct' with the received ciphertext ct helps evaluate the validity of ct (Line 21). For a valid ciphertext, comparison is successful with an overwhelming probability, and as a result, a valid shared secret K dependent upon the pre-shared secret $\bar{K}'$ and the received ciphertext ct' is generated (Line 24). However, for an invalid ciphertext, comparison fails with an overwhelming probability, resulting in generation of a pseudo-random secret K, using a pseudo-random value z and the received ciphertext ct' (Line 24). This provides strong theoretical security guarantees against chosen-ciphertext attacks which are possible over IND-CPA secure PKE/KEMs.

2.5 Dilithium

Dilithium is a lattice-based signature scheme secure, whose security is based on the Module LWE (M-LWE) and Module SIS (M-SIS) problem [20]. Dilithium operates over the module $R_q^{k \times \ell}$ with $(k, \ell) > 1$ where $R_q = \mathbb{Z}[x]/(x^n + 1)$, $n = 256$ and $q = 2^{23} - 2^{17} - 1$. This choice of parameters allows the use of NTT for polynomial multiplication in R_q. Dilithium also comes in three security levels: Dilithium2 with $(k, \ell) = (4, 4)$ at NIST Level 2, Dilithium3 with $(k, \ell) = (6, 5)$ at NIST Level 3 and Dilithium5 with $(k, \ell) = (8, 7)$ at NIST Level 5. There are two variants of Dilithium: (1) Deterministic (2) Probabilistic/Randomized, which only subtly differ in the way randomness is used in the signing procedure. The signing procedure of the deterministic Dilithium does not utilize external randomness and can generate only a single signature for a given message. The randomized variant however utilizes external randomness and thus generates a different signature, for a given message in each execution.

Algorithm 2: FO transform of a CPA-secure Kyber PKE into a CCA-secure Kyber KEM

procedure CCA.KeyGen
  $z \leftarrow \{0,1\}^{256}$
  $(pk, sk') \leftarrow$ CPA.KeyGen()
  $sk = (sk' \| \mathcal{H}(pk) \| z)$
  Return (pk, sk)
end procedure

procedure CCA.Encaps(pk)
  $m \leftarrow \{0,1\}^{256}$
  $m = \mathcal{H}(m)$
  $(\bar{K}, r) = \mathcal{G}(m \| \mathcal{H}(pk))$
  $ct =$ CPA.Encrypt(pk, m, r)
  $K = \mathsf{KDF}(\bar{K} \| \mathcal{H}(c))$
  Return (ct, K)
end procedure

procedure CCA.Decaps(sk, ct)
  $(pk, \mathcal{H}(pk), z) \leftarrow$ UnpackSK(sk)
  $m' =$ CPA.Decrypt(sk, ct)
  $(\bar{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
  $T = \bar{K}'$
  $ct' =$ CPA.Encrypt(pk, m', r')
  if (CompareCT$(ct', ct) == 0$) **then**
    $T = z$ ▷ Ciphertext Comparison Failure
  end if
  Return $K = \mathsf{KDF}(T \| \mathcal{H}(ct'))$
end procedure

2.5.1 Algorithmic Description. Refer Alg.3-4 for a simplified description of the key generation, signing and verification procedures of Dilithium. The functions Sample_U, Sample_B and Expand perform the same functions as in Kyber, albeit with different parameters. Dilithium also uses a number of rounding functions such as Power2Round, HighBits, LowBits, MakeHint and UseHint, whose details can be found in [20]. The key generation procedure simply involves generation of an LWE instance $\mathbf{t}$ (Line 4). Subsequently, the LWE instance is split into higher and lower order bits $\mathbf{t_1}$ and $\mathbf{t_0}$ respectively (Line 5), where $\mathbf{t_1}$ forms part of the public key, while $\mathbf{t_0}$ becomes part of the secret key.

The signing procedure of Dilithium is based on the "Fiat-Shamir with Aborts" framework where the signature is repeatedly generated and rejected until it satisfies a given set of conditions[30]. The message m is first hashed with a public value tr to generate μ (Line 11). The abort loop (Line 18-36) starts by generating an ephemeral nonce $\mathbf{y} \in R_q^\ell$, using a seed ρ. For the deterministic variant, the seed ρ is obtained by hashing μ with a secret nonce K (Line 14), while the probabilistic variant randomly samples the seed ρ from a uniform distribution (Line 16). This is the only differentiator between the two variants. The nonce $\mathbf{y}$ along with the public key component $\mathbf{A}$ is then used to calculate a sparse challenge polynomial $\mathbf{c} \in R_q$ (Line 22), whose 60 coefficients are either ± 1, while the other 196 coefficients are 0. Subsequently, the challenge $\mathbf{c}$, nonce $\mathbf{y}$ and secret $\mathbf{s}_1$, are used to compute the primary signature component $\mathbf{z}$ (Line 24). Then, a hint vector $\mathbf{h}$ is generated and output as part of the signature σ. The abort loop contains several conditional

checks (Line 26, 31), which should be simultaneously satisfied to terminate the abort loop and generate the signature $\sigma = (\mathbf{z}, \mathbf{h}, c)$.

The verification procedure utilizes the signature σ and the public key pk to recompute the challenge polynomial $\bar{c}$ (Line 38), which is then compared with the received challenge $\mathbf{c}$, along with other checks (Line 6 in Alg.4). If all the checks are satisified, then the verification is successful, else it is a failure. While we have only presented a simplified description of the Dilithium signature scheme, we refer the reader to [20] for a detailed description of the same.

Algorithm 3: Dilithium Signature scheme (Simplified)

1: **procedure** KeyGen
2: $(seed_A, seed_S, K) \in \mathcal{B} \leftarrow \mathsf{Sample}_U(); \mathbf{s_1}, \mathbf{s_2} \in (R_q^\ell \times R_q^k) \leftarrow \mathsf{Sample}_B(seed_S)$
3: $\mathbf{A} \in R_q^{k \times \ell} \leftarrow \mathsf{Expand}(seed_A)$
4: $\mathbf{t} = \mathbf{A} \cdot \mathbf{s_1} + \mathbf{s_2}$ ▷ Generate LWE instance $\mathbf{t}$
5: $(\mathbf{t_1}, \mathbf{t_0}) \leftarrow \mathsf{Power2Round}(\mathbf{t})$ ▷ Split $\mathbf{t}$ as $\mathbf{t_1} \cdot 2^d + \mathbf{t_0}$
6: $tr \in \mathcal{B} \leftarrow \mathcal{H}(seed_A \| \mathbf{t}_1)$
7: $pk = (seed_A, \mathbf{t_1}), sk = (seed_A, K, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0})$
8: **end procedure**

9: **procedure** Sign(sk, M)
10: $\hat{\mathbf{A}} \in R_q^{k \times \ell} \leftarrow \mathsf{Expand}(seed_A)$
11: $\mu \in \{0,1\}^{512} \leftarrow \mathcal{H}(tr \| M)$ ▷ Hash m with public value tr
12: $\kappa \leftarrow 0$; $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
13: **if** Deterministic **then**
14: $\rho \in R_q^\ell \leftarrow \mathcal{H}(K \| \mu)$ ▷ Generate seed ρ using message and secret seed K
15: **else**
16: $\rho \in R_q^\ell \leftarrow \mathsf{Sample}_U()$ ▷ Generate uniform seed ρ
17: **end if**
18: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ Start of Abort Loop
19: $\mathbf{y} \leftarrow \mathsf{Sample}_Y(\rho \| \kappa)$
20: $\hat{\mathbf{y}} = \mathsf{NTT}(\mathbf{y})$ ▷ $\mathsf{NTT}(y)$
21: $\mathbf{w} \leftarrow \mathsf{INTT}(\hat{\mathbf{A}} \circ \hat{\mathbf{y}})$; $\mathbf{w_1} \leftarrow \mathsf{HighBits}(\mathbf{w})$ ▷ $\mathsf{w}_1 = \mathsf{HighBits}(\mathbf{A} \cdot \mathbf{y})$
22: $\mathbf{c} \in R_q \leftarrow \mathcal{H}(\mu \| \mathbf{w_1})$ ▷ Generate Sparse Challenge c
23: $\hat{\mathbf{c}} = \mathsf{NTT}(c)$ ▷ $\mathsf{NTT}(\mathbf{c})$
24: $\mathbf{z} = \mathsf{INTT}(\hat{\mathbf{c}} \circ \hat{\mathbf{s}}_\mathbf{1}) + \mathbf{y}$ ▷ $\mathbf{z} = \mathbf{s_1} \cdot \mathbf{c} + \mathbf{y}$
25: $\mathbf{r}_0 = \mathsf{LowBits}(\mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2)$
26: **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then** ▷ Conditional Checks
27: $(\mathbf{z}, \mathbf{h}) = \perp$
28: $\kappa = \kappa + 1$
29: **else**
30: $\mathbf{h} = \mathsf{MakeHint}(-\mathbf{c} \cdot \mathbf{t}_0, \mathbf{w} - \mathbf{cs}_2 + \mathbf{c} \cdot \mathbf{t}_0, 2\gamma_2)$
31: **if** $\|\mathbf{c} \cdot \mathbf{t}_0\|_\infty \geq \gamma_2$ or #1's in $\mathbf{h} > \omega$ **then** ▷ Conditional Checks
32: $(\mathbf{z}, \mathbf{h}) = \perp$
33: $\kappa = \kappa + 1$
34: **end if**
35: **end if**
36: **end while**
37: $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$
38: **end procedure**

Algorithm 4: Dilithium Signature scheme (Simplified)

procedure VERIFY($pk, M, \sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$)
  $\mu \in \{0,1\}^{512} \leftarrow \mathcal{H}(tr \| M)$
  $\hat{\mathbf{c}} = \mathsf{NTT}(\mathbf{c})$
  $\mathbf{w}_1' := \mathsf{UseHint}(\mathbf{h}, \mathbf{A} \cdot \mathbf{z} - \mathsf{INTT}(\hat{\mathbf{c}} \circ \hat{\mathbf{t}}_\mathbf{1} \cdot 2^d, 2\gamma_2)$
  $\bar{\mathbf{c}} = \mathcal{H}(\mu, \mathbf{w}_1')$
  if ($\bar{\mathbf{c}} == \mathbf{c}$) and (norm of $\mathbf{z}$ and $\mathbf{h}$ are valid) **then**
    Return Pass
  else
    Return Fail
  end if
end procedure

3 SIDE-CHANNEL ATTACKS ON KYBER KEM

Given the variety of SCA/FIA performed on Kyber, we attempt to classify reported attacks along two orthogonal axes - (1) Based on the goal of the attacker and (2) Based on attacker's access to inputs/outputs of the target.

Based on the goal of the attacker: Along this axis, we classify SCA into two categories: (1) *Message Recovery* Attacks and (2) *Key Recovery* Attacks. Message recovery attacks attempt to recover the message m from a valid ciphertext ct, corresponding to a valid key exchange between two legitimate parties. Knowledge of m leads to recovery of the shared secret or session key K, thereby compromising the confidentiality of the corresponding targeted session. On the other hand, key recovery attacks attempt to recover the long term secret key sk manipulated in the decapsulation procedure. Recovery of sk leads to recovery of all the session keys K derived using the recovered sk. Thus, key recovery attacks are more attractive for an attacker, especially when the same key sk is used for multiple key exchanges (i.e.) use of static keys. However, in an epehemeral setting, where sk is refreshed for every key-exchange, message recovery attacks are as effective as key recovery attacks.

Based on attacker's access to target's input/output: Along this axis, we classify attacks into two categories: (1) *Known Ciphertext* Attacks (KCA) and (2) *Chosen Ciphertext* Attacks (CCA). *Known Ciphertext Attacks* are those which assume that the attacker only has knowledge of the ciphertexts (i.e.) ciphertexts generated from the target encapsulation procedure or ciphertexts submitted to the target decapsulation procedure. In this setting, the attacker can only passively observe the target device, while not being able to establish communication with it. For convenience and simplicity, we also include attacks on the key-generation procedure in this category, where the attacker only has access to the generated public keys. On the other hand, *Chosen Ciphertext Attacks* are those which assume the adversary's capability to establish communication with the target. This is applicable in a setting where the attacker can query the target decapsulation device with chosen ciphertexts of his/her choice.

We adopt the following nomenclature to categorize the different types of SCA/FIA on Kyber KEM: ⟨SCA/FIA⟩_⟨Attacker Goal⟩_⟨Access to Inputs and Outputs⟩_⟨Attack Name⟩ where Attack Label refers to the unique label given to identify the different attacks discussed in this paper. In the following, we present a brief survey of the various types of side-channel attacks mounted on structured lattice-based KEMs, with main focus on attacks that are applicable to Kyber. We utilize as reference, the algorithms of CPA secure Kyber PKE in Alg.1 and CCA secure Kyber KEM in Alg.2.

3.1 Key Recovery Attacks - Known Ciphertext Scenario (SCA_KR_KCA)

Key recovery attacks in the KCA setting typically target leakage from operations that directly manipulate the secret module $\mathbf{s}$ within the decapsulatiom procedure. In this respect, the NTT based polynomial multiplication used in the decryption procedure, has been shown to be exploitable for key recovery. Primas *et al.* [41] presented a template style SCA targeting the NTT, relying on Soft-Analytical Side-Channel Attack (SASCA) based techniques [54] for key recovery. They were able to recover the entire key in a single side-channel trace (Power/EM) from a Ring-LWE scheme running on an ARM Cortex-M4 microcontroller. They target leakage from the operation $\mathsf{INTT}(\hat{\mathbf{u}}' \circ \hat{\mathbf{s}})$ (Line 26 in Alg.1) in the decryption procedure. The goal is to recover its input (i.e.) $\hat{\mathbf{u}}' \circ \hat{\mathbf{s}}$ which leads to recovery of the secret key $\mathbf{s}$.

The attack works in two phases. Firstly, a profiling phase is used to construct templates for intermediate operations such as modular multiplication, loads and stores of the inputs and outputs of every butterfly operation in the INTT. In the attack phase, these templates are matched with the corresponding segments in the attack trace, and the results are combined using the well-known Belief Propagation (BP) algorithm [38] to recover the secret key. There are a few downsides of this attack - (1) Requirement of an extensive profiling phase (with more than 100 million templates as shown in [41]) (2) Detailed knowledge of the INTT's implementation and (3) Requirement of relatively high SNR for successful key recovery. We refer to the attacks targeting the NTT using the label NTT_Leakage.

There have also been reported side-channel attacks on other types of polynomial multipliers such as the Toom-Cook, Karatsuba and the school-book polynomial multipler [6, 13, 33], used in other lattice-based KEMs such as Saber and NTRU. However, these attacks are not relevant to Kyber as use of NTT is included in the algorithmic specification of Kyber [5].

3.2 Message Recovery Attacks - Known Ciphertext Scenario (SCA_MR_KCA)

While the aforementioned attack focussed on key recovery, message recovery is also possible in a KCA setting. In this setting, the attacker only has access to a single execution of the encapsulation/decapsulation procedure to perform message recovery from the target ciphertext ct. These attacks can be split into two categories.

3.2.1 Targeting Message Encoding and Decoding Operation: Message recovery attacks have predominantly targeted two operations that directly manipulate the sensitive message m (i.e.) Encode operation in encryption (Line 20 of CPA.Encrypt procedure in Alg.1) and Decode operation in decryption (Line 27 of CPA.Decrypt procedure in Alg.1). Both the encoding and decoding operations manipulate the message one bit at a time, and this bitwise manipulation of the sensitive message serves as the primary source of leakage for several reported attacks [3, 42, 51]. The first such attack was demonstrated by Amiet *et al.* [3] targeting the message encoding operation in NewHope KEM, a Ring-LWE based KEM on the ARM Cortex-M4 microcontroller. The difference in Hamming Weight of the message polyomial coefficients (i.e.) $\mathbf{m}[i] = \lceil q/2 \rfloor$ for $m_i = 1$ and $\mathbf{m}[i] = 0$ for $m_i = 0$ could be easily distinguishable through SCA, thereby enabling complete recovery of individual messge bits in a single trace. Subsequently, Sim *et al.* [51] generalized the attack technique to target all lattice-based KEMs in the NIST standardization process including Kyber KEM on the same platform.

Subsequently, Ravi *et al.* [42] presented novel attacks that exploit leakage from the message decoding operation in the decryption procedure for message recovery. Though they demonstrated presence of leakage from individual message bits, they were only able to obtain a success rate of 81% for recovering single message bytes of Kyber, while a setup with higher SNR could potentially perform perfect single trace message recovery. We refer to these attacks targeting the encoding/decoding procedure together using the label Encode_Decode_Leakage.

3.2.2 Targeting NTT operation: Pessl *et al.* [39] demonstrated that the NTT operation can also be targeted for message recovery. Their idea was to recover the input to the NTT instance over the ephemeral secret **r** (Line 17) whose knowledge can be used to recover the message m from the ciphertext ct. They also proposed significant improvements to the original attack of Primas *et al.* [41], by reducing the number of templates from 1 million to just 213 templates, while also presenting several improvements such as relying on an improved BP algorithm for message recovery. This attack was also shown to be applicable to masking countermeasures, albeit in the presence of a high SNR. We use the same label NTT_Leakage to refer to the aforementioned attack targeting the NTT in a KCA setting for message recovery.

In the following, we shift focus to key recovery and message recovery attacks that can be performed in a chosen-ciphertext setting.

3.3 Key Recovery Attacks - Chosen Ciphertext Scenario (SCA_KR_CCA)

Kyber KEM is IND-CCA secure, and therefore enjoys concrete theoretical security guarantees against classical chosen-ciphertext attacks. This is primarily due to the attacker's inability to access any information about sensitive intermediate variables in the decapsulation procedure for malicious and handcrafted chosen ciphertexts. However, several works have shown that the attacker can craft chosen-ciphertexts, which when decapsulated have the ability to amplify secret-dependent leakage, from several operations within the decapsulation procedure. In the following, we discuss the different types of CCAs that have been mounted for key recovery.

3.3.1 Side-Channel Oracle Based Attacks: This forms the major category of key recovery attacks in the CCA setting. Their modus operandi is given as follows: The attacker queries the decapsulation procedure with handcrafted ciphertexts. These ciphertexts are crafted such that the decrypted message m' is very closely related to a targeted portion of the secret key, or in a few cases, the entire secret key. The attacker utilizes leakage from operations processing the decrypted message to recover the same, thereby realizing a practical side-channel *oracle*. Such information obtained over several carefully crafted ciphertexts, reveals the full secret key. Following are the three major sub-categories of side-channel oracle based CCAs.

Plaintext-Checking Oracle-Based SCA: The core idea of these attacks is to construct ciphertexts, so as to restrict the number of possibilites of the decrypted message. Moreover, the value of the decrypted message also depends upon a single targeted coefficient of the secret key, for the chosen-ciphertexts. Side-channel leakage from operations processing the message are used to instantiate a *Plaintext-Checking* (PC) oracle for key recovery. D'Anvers *et al.* [18] presented PC oracle-based SCA on PQC schemes such as LAC and RAMSTAKE exploiting the timing-side channel from non-constant time error correcting codes. Subsequently, Ravi *et al.* [49] generalized the attack using the EM side-channel to all LWE/LWR-based KEMs in the second round of the NIST process, including Kyber KEM. We now briefly the same attack on Kyber KEM. Referring to Alg.1, the attacker chooses a very sparse ciphertext $ct = (\mathbf{u}, \mathbf{v}) \in (R_q^k \times R_q)$ as follows:

$$\mathbf{u}_i = \begin{cases} U \cdot x^0 & \text{if } i = 0, \\ 0 & \text{if } 1 \leq i \leq k-1 \end{cases} \tag{4}$$

$$\mathbf{v} = V \cdot x^0 \tag{5}$$

where $(U, V) \in \mathbb{Z}^+$. For this chosen-ciphertext, each bit of the decrypted message m' (i.e.) m'_i for $i \in [0, n-1]$ is given as:

$$m'_i = \begin{cases} \text{Decode}(V - U \cdot \mathbf{s}_0[0]), & \text{if } i = 0 \\ \text{Decode}(-U \cdot \mathbf{s}_0[i]), & \text{for } 1 \leq i \leq n-1 \end{cases} \tag{6}$$

Thus, every bit m'_i is only dependent on a single corresponding secret coefficient of $\mathbf{s}_0$ (i.e.) $\mathbf{s}_0[i]$. The attacker can chooses tuples (U, V) such that:

$$m'_i = \begin{cases} \mathcal{F}(\mathbf{s}_0[0]), & \text{if } i = 0 \\ 0, & \text{for } 1 \leq i \leq n-1 \end{cases} \tag{7}$$

Now, m' can only take two possible values (i.e.) $m' = 0/1$, whose value depends upon a single secret coefficient $\mathbf{s}_0[0]$. Thus, $m' = 0/1$ for different tuples (U, V) can be used as a binary distinguisher for every possible candidate of $\mathbf{s}_0[0]$. In the similar manner, complete secret key can be recovered one at a time, by appropriate modifying the chosen-ciphertexts. Instantiating an oracle to distinguish $m' = 0/1$ through power/EM side-channels can be done using a single trace, as shown by Ravi *et al.* [49]. This is because a single-bit difference in m' uniformly randomizes all subsequent operations after decryption due to the use of hash functions in the decapsulation procedure (Line 18 in Alg.2).

Since the attack recovers binary information in every query, full key recovery can be done in $\approx 2k - 4k$ queries across all the parameter sets of Kyber KEM. One of the main advantages is that, the attack can exploit leakage from the entire re-encryption procedure (Line 19), and thus can work in a low SNR setting and low-cost SCA equipment. We refer to this attack using the label PC_Oracle.

Decryption-Failure Oracle-Based SCA: This category of attacks work by querying the decapsulation device with carefully perturbed ciphertexts, such that the induced decryption failures due to the perturbations, depend upon the secret key. A side-channel oracle that is able to detect the decryption failures can therefore recover the secret key. The first such SCA based decryption failure oracle attack was proposed by Guo *et al.* [26] on Frodo KEM, exploiting the non-constant time execution of the ciphertext comparison operation to detect decryption failures. Subsequently, Bhasin *et al.* [11] generalized the attack to Kyber KEM and demonstrated that power/EM side-channel leakage from the ciphertext comparison block can be used to detect decryption failures for key recovery.

We briefly explain the same attack on Kyber KEM. The attacker generates a valid ciphertext $ct = (\mathbf{u}, \mathbf{v})$ for a message m and adds single coefficient errors to the second component $\mathbf{v}$ (e.g.) $\mathbf{v}^* = \mathbf{v} + e \cdot x^0$ (adding error to the first coefficient) where $e \in \mathbb{Z}$. This has the effect of perturbing $\mathbf{m'}[0]$, by increasing the noise $\mathbf{d}[0]$ by e (i.e.) $\mathbf{d'}[0] = \mathbf{d}[0] + e$. As long as $\|\mathbf{d'}[0]\| < q/4$, it results in correct decryption (i.e.) $m' = m$. However, if the perturbation results in $\|\mathbf{d'}[0]\| > q/4$, then it flips the corresponding message bit m'_0, resulting in a decryption failure. The size of e that triggers a decryption failure provides information about the original noise $\mathbf{d}[0]$. Since the noise $\mathbf{d}$ in the message polynomial is linearly dependent on the secret $\mathbf{s}$ (Eqn.3), enough information about $\mathbf{s}$ results in full key recovery.

Several recent works [21, 28] have shown that simply recovering the sign of $\mathbf{d}[0]$ (i.e.) $\mathbf{d}[0] > 0$ or $\mathbf{d}[0] < 0$ for several valid ciphertexts ($\approx 5k - 7k$), is sufficient to fully recover the secret key. A decryption failure can be easily identified using side-channel leakage from any operation within the re-encryption procedure (Line 18 in Alg.2) as well as the ciphertext comparison operation (Line 21). Bhasin *et al.* [11] and D'Anvers *et al.* [21] particularly targeted leakage from the ciphertext comparison operation (Line 21) and were also able to break flawed masked implementations of the ciphetext comparison operation [7, 37]. We refer to these attacks using the label DF_Oracle.

Full-Decryption Oracle-Based SCA: While the aforementioned PC_Oracle and DF_Oracle attacks only extract binary information in every query, it raises a natural question if it is possible to instantiate a more powerful oracle to obtain more than binary information about the decrypted message. In this respect, Xu *et al.* [56] proposed a novel techinque to construct chosen ciphertexts which simultaneously provide 256 bits of information about the secret key $\mathbf{s}$. They construct $ct = (\mathbf{u}, \mathbf{v}) \in (R_q^k \times R_q)$ such that

$$\mathbf{u}_i = \begin{cases} U \cdot x^0 & \text{if } i = 0, \\ 0 & \text{if } 1 \leq i \leq k-1 \end{cases} \tag{8}$$

$$\mathbf{v} = V \cdot (\sum_{i=0}^{i=n-1} x^i) \tag{9}$$

where $(U, V) \in \mathbb{Z}^+$. The attacker can choose tuples (U, V) such that the decrypted message is nothing but

$$m'_i = \begin{cases} \mathcal{F}(\mathbf{s}_0[i]), & \text{if } 0 \leq i \leq n-1 \end{cases} \tag{10}$$

where every message bit m'_i is dependent upon the corresponding secret coefficient of $\mathbf{s}_0$ (i.e.) $\mathbf{s}_0[i]$. Moreover, attacker can choose (U, V) such that every message bit m'_i uniquely identifies corresponding secret coefficient $\mathbf{s}_0[i]$. In this manner, the attacker has effectively parallelized the PC_Oracle attack. Since key recovery requires access to the complete decrypted message (i.e.) *full decryption* oracle, Xu *et al.* [56] proposed to exploit leakage from the message encoding operation during re-encryption (Line 20 in Alg.2), which can be used to recover 256 bits in a single trace.

Thus, full key recovery is possible in only 6 queries for Kyber512. Similarly, Ravi *et al.* [42] and Ngo *et al.* [35, 36] demonstrated the possibility of exploiting leakage from the message decoding operation in a chosen-ciphertext setting for full key recovery, in approximately $6 - 20$ traces from schemes such as Kyber and Saber. We refer to these attacks using the label FD_Oracle. In essence, the PC_Oracle, DF_Oracle FD_Oracle attacks demonstrate that an attacker can utilize chosen-ciphertexts to extract leakage from different operations within the decapsulation procedure for key recovery. Table 1 lists side-channel assisted CCAs on IND-CCA secure LWE/LWR-based schemes by their oracle types.

Table 1. Classification of side-channel assisted CCAs on IND-CCA secure LWE/LWR-based schemes according to oracle type. The anchor variable is denoted by anchor and m_x, with or without subscript, is the decrypted message.

Type of Oracle	Oracle Response
Plaintext-Checking (PC)	$m' \in \{m_0, m_1\}$
Decryption-Failure (DF)	$m' \in \{m_{\text{valid}}, m_{\text{invalid}}\}$
Full-Decryption (FD)	$m' = m$

3.3.2 Targeting NTT in a CCA Scenario. While leakage from NTTs have been used for message recovery and key recovery in a KCA setting in [39, 41], these attacks rely on relatively low-noise measurements for successful key recovery. Specifically, these attacks could only tolerate a noise with standard deviation σ in the range $0.5 - 0.7$. Recently, Hamburg *et al.* [27] demonstrated that the sensitivity of these attacks to SNR can be significantly improved in a chosen-ciphertext setting. Their idea was to craft chosen-ciphertexts so as to feed a sparse input ($\hat{\mathbf{u}}' \circ \hat{\mathbf{s}}$) to the INTT instance in the decryption procedure (Line 26 in Alg.1). This reportedly improves the effectiveness of the BP algorithm, by allowing more noise in the measurements, even when targeting masked implementations. They demonstrate a range of key

recovery attacks with trace complexity ranging from k to $2k$ where k is the dimension of the module in Kyber KEM ($k = \{2, 3, 4\}$). The improved attack can tolerate much more noise with $\sigma \leq 2.2$, thereby demonstrating significant improvement in NTT attacks when performed in a chosen-ciphertext setting. We refer to the attacks targeting the NTT using the label NTT_Leakage.

3.4 Message Recovery Attacks - Chosen Ciphertext Scenario (SCA_MR_CCA)

We recall that the message recovery attacks targeting the message encoding and decoding operations are capable of recovering the entire message in a single trace in a known-ciphertext setting [3, 42, 51]. However, these attacks can be thwarted using a simple shuffling countermeasure, which randomize the order of encoding/decoding of the single message bits. While shuffling does not remove the source of side-channel leakage, the attacker cannot recover the correct order of message bits, thereby thwarting single-trace message recovery. However, Ravi *et al.* [42] showed that an attacker can break the shuffling countermeasure in a chosen-ciphertext setting, utilizing the *ciphertext malleability* property of LWE/LWR-based schemes.

We briefly describe their attack on the shuffling countermeasure, which recovers the message one bit at a time. Given a target ciphertext $ct = (\mathbf{u}, \mathbf{v})$, the attacker first submits ct to the decapsulation procedure to recover the individual message bits of m' through side-channels and subsequently computes its Hamming Weight (HW). Subsequently, the attacker submits a perturbed ciphertext $ct^* = (\mathbf{u}, \mathbf{v} + q/2 \cdot x^0)$ (i.e.) $q/2$ added to the first coefficient of $\mathbf{v}$. This has the effect of flipping the first message bit m'_0, resulting in a perturbed message m''. The difference in the HW of m' and m'' (increase or decrease) can be used to recover the value of the flipped message bit m_0. In this way, complete message recovery is possible in 257 queries for Kyber KEM. Recently, Ngo *et al.* [36] extended the same attack to also break the combined shuffling and masking countermeasure for the message decoding operation in Saber. Here again, we can clearly observe the increase in attacker's capability to perform improved attacks in a chosen-ciphertext setting , compared to the known-ciphertext setting. We refer to these attacks targeting the protected message encoding/decoding procedure using the label Protected_Encode_Decode_Leakage.

4 FAULT-INJECTION ATTACKS ON KYBER KEM

In this section, we present a brief survey of the fault injection attacks on structured lattice-based KEMs, with main focus on attacks that are applicable to Kyber.

4.1 Key Recovery and Message Attacks - Known Ciphertext Scenario (FIA_KR_KCA and FIA_MR_KCA)

This category covers attacks on the key-generation and encapsulation procedure, where the attacker can only observe faulty outputs from the target. Ravi *et al.* [48] proposed the first practical fault attack applicable to lattice-based KEMs such as Kyber, NewHope and Frodo. Their attack stems from the observation that the seed used to sample the secret and errors for the LWE instances only differ by a single byte (i.e.) $\mathbf{s}$ and $\mathbf{e}$ are sampled from the same seed $seed_B$, but with different nonces $coins_s$ and $coins_e$ which only differ by a single byte (Line 5,6 in CPA.KeyGen of Alg.1). The same is also applicable to the encryption procedure (Line 14,15). Thus, the attacker can use faults to force nonce reuse (i.e.) $coins_s = coins_e$, to create LWE instances of the form, $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{s} = \mathbf{A} \cdot (\mathbf{s} + 1)$, that can be trivially solved using Gaussian elimination. The authors demonstrated practicality of nonce-reuse using Electromagnetic Fault Injection (EMFI) on the ARM Cortex-M4 microcontroller. While the attack leads to full key recovery and message recovery in a Man In The Middle (MITM) setting, it requires to inject multiple targeted faults in the key-generation and encapsulation procedure for practical attacks. We refer to this attack using the label Nonce_Reuse.

Valencia *et al.* [53] performed a more general study of the susceptibility of CPA secure LWE/LWR-based schemes to fault attacks. They propose a variety of attacks targeting several operations within key-generation, encryption and decryption procedures. However, their simulated attacks consider fault models such as zeroization of entire polynomials, which is hard to achieve in practice. Moreoever, their attack on the decryption procedure only applies to CPA secure Kyber PKE, and thus is not applicable to CCA secure Kyber KEM.

In the following, we discuss key recovery attacks targeting the decapsulation procedure with chosen-ciphertexts, which form the major category of FIA on Kyber KEM.

4.2 Key Recovery Attacks - Chosen Ciphertext Scenario (FIA_KR_CCA)

One of the main challenges of targeting the decapsulation procedure through faults is that, it contains an inherent protection against faults (i.e.) FO transform to detect invalid ciphertexts with a very high probability. This appears as a natural protection, and therefore presents a significant challenge to perform FIA.

4.2.1 Targeting Ciphertext Equality Check. One obvious target within the decapsulation procedure is to simply skip the ciphertext equality check through faults (Line 21 in Alg.2). Recently, Xagawa *et al.* [55] surveyed optimized software implementations of several PQC schemes on the ARM Cortex-M4 microcontroller and identified that the CCA security of several schemes including Kyber can be easily broken through a single targeted fault. We analyzed the implementation of ciphertext equality check operation within the software implementation of Kyber KEM from the *pqm4* library [29]. An array T holds the sensitive pre-shared secret $\bar{K}'$ derived from the decrypted message m' after decryption (Line 19 of CCA.Decaps in Alg.2). If ciphertext comparison fails (invalid/malicious ciphertext), a pseudo-random value z is written into T susing a conditional move operation (Line 22). Subsequently, T is used to derive the final shared secret K (Line 24).

Thus, the decapsulation procedure writes the sensitive pre-shared secret onto T (assuming successful decapsulation), before checking the validity of the ciphertext. Thus, simply skipping the subsequent conditional move operation ensures that the sensitive pre-shared secret $\bar{K}'$ is used to generate the shared secret K, even upon failure of the ciphertext comparison operation. Xagawa *et al.* showed that the aforementioned vulnerability can be exploited through simple clock glitches and can subsequently lead to key recovery in a few thousand queries, through a chosen-ciphertext attack [49]. We refer to this attack using the label Skip_CT_Compare.

4.2.2 Fault Assisted CCAs. Barring the ciphertext equality check, there are no other trivial fault targets within the decapsulation procedure. However, Pessl and Prokop [40] recently proposed the first generic fault assisted CCA, which works by injecting targeted faults within the message decoding operation within decryption, such that the resulting success/failure of decapsulation can be used to infer critical information about the secret key.

We briefly describe the main idea of their attack. The attacker submits a valid ciphertext ct for decapsulation, and injects a single fault to skip the addition with $q/2$ during decoding of a single message polynomial coefficient $\mathbf{m'}[i]$ (Refer Fig.1 for the message decoding routine). This has an indirect effect of perturbing $\mathbf{m'}[i]$ approximately by $q/4$. This results in a flip of m'_i (decapsulation failure) only when the corresponding coefficient of the noise component $\mathbf{d}[i] < 0$. However, there is no change in the m'_i when $\mathbf{d}[i] \geq 0$. This helps the attacker build a single linear inequality using $\mathbf{d}[i]$, and the attacker who is able to build $5k - 7k$ such linear inequalities can perform full key recovery over Kyber KEM. While the attack was demonstrated using clock glitching on the ARM Cortex-M4 microcontroller, the attack requires to still inject a targeted skipping fault in the message decoding procedure. Thus, this attack can be thwarted by simply shuffling the message decoding operation.

Subsequently, Hermelink *et al.* [28] proposed improvements to the attack of Pessl and Prokop [40], by adopting a slightly different approach. Instead of using a valid ciphertext ct, they propose to submit perturbed ciphertexts, such that a single coefficient of the second ciphertext component $\mathbf{v}[i]$ is perturbed by $q/4$. Upon submitting the perturbed ciphertext, a fault is injected after decryption, to correct the single-bit perturbation in the ciphertext stored in memory. If the introduced perturbation resulted in correct decryption ($\mathbf{d}[i] \geq 0$), then the injected fault corrects the perturbation in the ciphertext ensuring successful decapsulation. However, if the initial perturbation resulted in a decryption failure ($\mathbf{d}[i] < 0$), then it results in decapsulation failure, even after correcting the perturbation in the stored ciphertext through faults. This information obtained about $\mathbf{d}$ over $5k - 7k$ such queries can recover the full secret key.

Unlike the attack of Pessl and Prokop, the attack of Hermelink *et al.* [28] does not have any timing constraints for fault injection, as it only needs to inject a bit-flip fault in memory, anytime between the decryption and ciphertext comparison operation. However, injecting precise single bit-flip faults in memory requires detailed information about the target device as well as the implementation, and an extensive profiling of the target device. More recently, Delvaux [19] improved the attack of Hermelink *et al.* [28] by expanding the attack surface to several operations within the decapsulation procedure, while also working with a variety of more relaxed fault models such as arbitrary bit flips, set-to-0 faults, random faults and instruction skip faults. However, attacks relying on a relaxed fault model could require more than $100k$ chosen-ciphertext queries for full key recovery, depending upon the practicality of the fault model. We refer to all the aforementioned attacks, together using the label Fault_Assisted_CCA.

Putting all the aforementioned SCA and FIA together, Please refer Fig.2 for compilation of the reported SCA and FIA on Kyber KEM, as well as those applicable to the same.

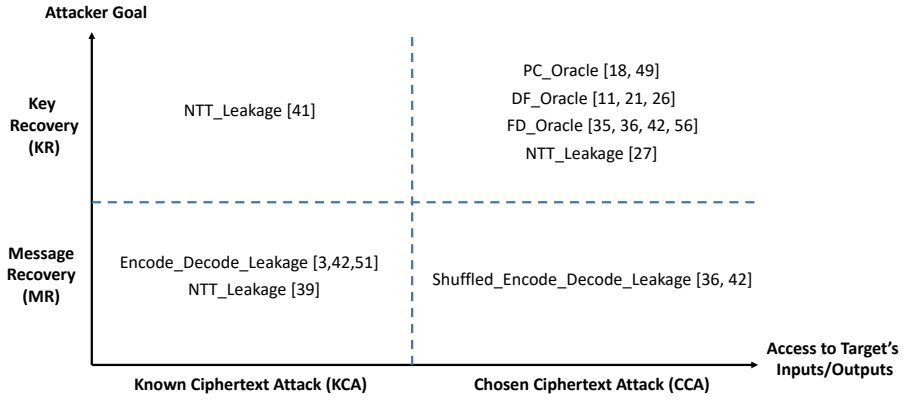
5 PROTECTING KYBER KEM AGAINST SCA/FIA

In the previous section, we have presented a detailed survey of the various SCA and FIA on Kyber KEM. In this section, we attempt to present a range of customized countermeasures that can be used to protect against the aforementioned attacks.

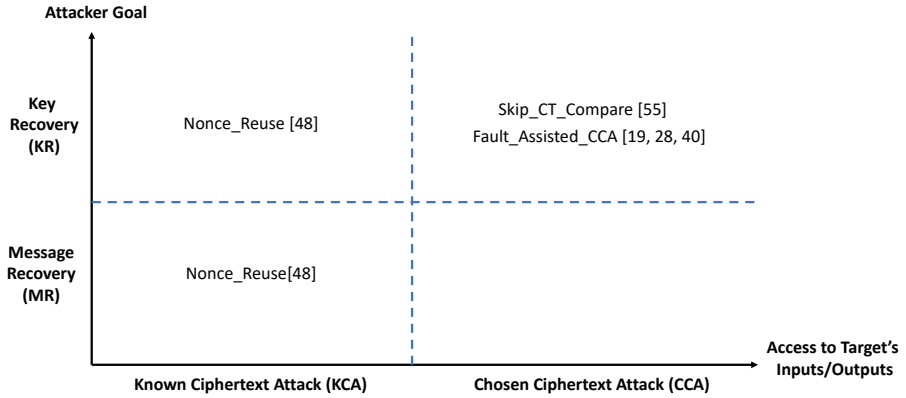
5.1 Protection Against SCA/FIA Assisted CCA

We observe that SCA and FIA performed in a chosen-ciphertext setting form the major category of attacks on Kyber KEM. With respect to protection against SCA, masking is widely considered to offer concrete protection, especially against multi-trace attacks. However, masking lattice-based schemes in practice has several shortcomings, which might affect its adoption in real-world applications. Firstly, masking the decapsulation procedure imposes a significant penalty on performance (speed), as clearly shown by several works [10, 14]. Bos *et al.* [14] papered a 3.5× increase in runtime for a first-order masked decapsulation for Kyber KEM on the ARM Cortex-M4, while Beirendonck *et al.* [10] papered a 2.5× increase in runtime for Saber KEM on the same platform.

Secondly, masking is not foolproof in practice, as shown by Ngo *et al.* [35] who proposed a side-channel attack on a first-order masked implementation of Saber KEM, which breaks with an incremental attacker's effort compared to attack on an unprotected implementation [42]. Moreover, Bhasin *et al.* [11] and D'Anvers *et al.* [21] have demonstrated practical attacks exploiting flaws in different masking schemes [7, 37] for the ciphertext comparison operation. Thus, designing a flawless masking scheme for different operations within the decapsulation procedure, appears to be tricky in practice. Thirdly, it is not clear which order of masking protection is required to achieve security in a given setting, especially given that the cost of masking significantly increases with order of protection. Finally, masking is a



(a) Side-Channel Attacks (SCA) on Kyber KEM



(b) Fault-Injection Attacks (FIA) on Kyber KEM

Fig. 2. Compilation of the reported SCA and FIA on Kyber KEM, as well as those applicable to the same.

SCA countermeasures, and thus does not offer protection against fault attacks (FIA_KR_CCA [40]), thus additional countermeasures are required for simultaneous protection against both SCA and FIA.

Thus, in this work, we propose an alternate approach to protect against SCA and FIA assisted CCA. Our approach relies on a *detection strategy*, to test/detect whether a received ciphertext is malicious. If detected as malicious, the target can simply reject the ciphertext and change/refresh the public-private key pair by re-running the key-generation procedure. An advantage of this approach is that upon detection, further exposure of the secret key is prevented. While such a protection is already available in the decapsulation procedure, in form of the FO transform, the minimum number of decapsulation failures that can be tolerated before refreshing the secret key is not clear. Moreover, the FO transform cannot distinguish between a genuinely invalid ciphertext (due to errors in communication) or a maliciously crafted ciphertext. Thus, we argue that a more concrete approach to identify malicious ciphertexts used for CCA is required. In the following, we propose two *detection* countermeasures against the proposed CCAs for Kyber KEM.

5.1.1 Ciphertext Sanity Check. A close observation of the ciphertexts used in the PC_Oracle-based CCA (Eqn.9) and FD_Oracle-based CCA (Eqn.9) reveal that most of the coefficients of the ciphertext have a fixed value of 0. However, the

coefficients of a valid ciphertext are uniformly distributed in the range $[0, q]$, given that both ciphertext components are essentially LWE instances. We therefore propose to perform a statistical analysis of the ciphertext, before being used within the decryption procedure. While this countermeasure was also proposed by Xu *et al.* [56], a concrete mathematical analysis and implementation of the same is not presented.

We choose to rely on the mean and standard deviation of the ciphertext coefficients, as the statistical measures to detect the skew in the received ciphertext. For a given polynomial $\mathbf{x} \in R_q$, we denote the mean (μ) and standard deviation (σ) of the coefficients of $\mathbf{x}$ as $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ respectively. We performed empirical simulations to calculate the mean and standard deviation of $\mu(\mathbf{u})$ and $\sigma(\mathbf{u})$ for single polynomials of the ciphertext component $\mathbf{u}$, as well as $\mu(\mathbf{v})$ and $\sigma(\mathbf{v})$ for the ciphertext component $\mathbf{v}$, corresponding to valid ciphertexts of Kyber KEM. Refer below for the obtained values for the mean and standard deviation of all 4 of the statistical metrics.

$$\begin{aligned}(\mu(\mu(\mathbf{u})), \sigma(\mu(\mathbf{u}))) &= (1663, 60) \\ (\mu(\sigma(\mathbf{u})), \sigma(\sigma(\mathbf{u}))) &= (959, 27)\end{aligned} \tag{11}$$

$$\begin{aligned}(\mu(\mu(\mathbf{v})), \sigma(\mu(\mathbf{v}))) &= (1560, 60) \\ (\mu(\sigma(\mathbf{v})), \sigma(\sigma(\mathbf{v})) &= (957, 27)\end{aligned} \tag{12}$$

Based on the standard deviation σ for each of these metrics, the designer can choose an acceptable range for each of these 4 metrics. For example, if a tail length of $6 \cdot \sigma$ is chosen, then the acceptable range for $\mu(\mathbf{u})$ is $[\mu(\mu(\mathbf{u})) + 6 \cdot \sigma, \mu(\mu(\mathbf{u})) - 6 \cdot \sigma]$. Smaller the acceptable range, higher is the possiblity of false positives (i.e.) detecting a valid ciphertext as invalid. However, a large acceptable range increases the chances of false negatives, thereby resulting in acceptance of skewed malicious ciphertext as valid.

We deduced through empirical simulations that a tail of length 6σ for both mean and standard deviation leads to a probability of $\approx 2^{-22}$ for rejection of a valid ciphertext. The implementor/designer can choose an appropriate range, based on the tolerance to allow false positives and rejection of valid ciphertexts. We henceforth refer to this as the CT_Sanity_Check countermeasure in this paper. One can also include other kinds of checks such as checking the number of zero coefficients in the received ciphertext, which can enhance confidence in the detection mechanism.

While this countermeasure is capable of detecting skewed ciphertexts, chosen-ciphertexts used in the DF_Oracle-based CCA [11] as well as those used in the Fault_Assisted_CCA attacks [19, 28, 40] are uniformly in random. This is because these attacks involve addition of small errors to a single coefficient of a valid ciphertext. This does not introduce a detectable skew in the coefficients, thereby defeating the CT_Sanity_Check the countermeasure. In the following, we propose a novel countermeasure that is also capable of defeating CCA utilizing chosen ciphertexts with uniformly random cofficients.

5.1.2 Message Polynomial Sanity Check. This countermeasure relies on analysing the coefficients of the noisy message polynomial $\mathbf{m'} = (\mathbf{v'} - \mathbf{u'} \cdot \mathbf{s})$ obtained during decryption of the received ciphertext ct (Line 26 of CPA.Decrypt procedure in Alg.1). For valid ciphertexts, we observe that the coefficients of the $\mathbf{m'}$ are distributed according to a very narrow Gaussian distribution near $q/2$ or 0 (i.e.) $\mathbf{m}[i] = q/2 \pm \delta$ for $m_i = 1$ and $\mathbf{m}[i] = 0 \pm \delta$ for $m_i = 0$

$$\mathbf{m}[i] = \begin{cases} q/2 \pm \delta & \text{if } m_i = 1, \\ 0 \pm \delta & \text{if } m_i = 0 \end{cases} \tag{13}$$

where $\delta \ll q \in \mathbb{Z}^+$. The span δ depends upon the distribution of the noise component $\mathbf{d}$ (Eqn.3). We performed empirical simulations and estimated a mean of 0 and a standard deviation $\sigma = 79$ for the noisy message polynomial coefficients clustered around 0 and $q/2$.

However, we observe that this distribution is not maintained in case of the DF_Oracle-based CCA and Fault_Assisted_CCA attacks. While the fault attack of Pessl *et al.* [40] adds $q/4$ to $\mathbf{m'}[i]$ through faults, the attacks of [11, 19, 28] explicitly add $q/4$ to the targeted coefficient of the valid ciphertext. Thus, all these attacks work by directly/indirectly adding $q/4$ to one of the targeted coefficients of the message polynomial $\mathbf{m'}[i]$. This ensures that atleast one message polynomial (i.e.) $\mathbf{m'}[i]$ is not within the expected range, corresponding to that of a valid ciphertext.

Based on this observation, we propose to test the distribution of the message polynomial coefficients for the received ciphertext. Let the acceptable range be $(q/2 \pm L \cdot \sigma)$ and $(0 \pm L \cdot \sigma)$ where $L \in \mathbb{Z}^+$ is left to the designer's choice. Larger the acceptable range $L \cdot \sigma$, smaller is the probability of flagging a valid ciphertext (false positive). However, choosing a smaller range raises the chances of missing detection of a malicious chosen-ciphertext. Thus, it is important to choose a conservative value for L for improved security. For $L = 6$ and we were not able to observe a false positive for more than 2^{25} valid decapsulations.

If there is atleast one coefficient outside this acceptable range, then we simply flag the ciphertext as valid, and refresh the public-private key pair. We observe that this countermeasure requires to decrypt atleast one chosen-ciphertext for successful detection, however the CCAs in interest require atleast a few tens to few thousand queries for key recovery. Thus, we argue that allowing a single decapsulation of the chosen-ciphertext is not useful for the attacker. We henceforth refer to this countermeasure as Message_Poly_Sanity_Check throughout this paper. It is important to note that, apart from the oracle-based attacks, this countermeasure is also capable of protection against the attack of Hamburg *et al.* [27], targeting the NTT leakage in a chosen-ciphertext setting.

5.2 Protecting Ciphertext Comparison

We propose two levels of protection for the ciphertext comparison operation, targeted by the Skip_CT_Compare attack of Xagawa *et al.* [55]. In the first level, we add protection against skipping the ciphertext comparison operation (Line 21 in Alg.2), for which we utilize a dynamic loop counter to keep track of the number of compared cipherext bytes in the ciphertext comparison operation. If the number of ciphertext bytes to be compared is d, then the loop counter l is initialized with a random value $k \cdot d$ where $k \in \mathbb{Z}^+$ is randomly chosen in every execution. Subsequently, l is decremented by k for every ciphertext byte compared. Thus, $l = 0$ after ciphertext comparison provides assurance that all bytes were compared. The use of such a dynamic loop counter (with a variable initial value), adds an extra layer of protection against higher order fault attacks that also attempt to fault the loop counter.

To protect against skipping of the conditional move operation (Line 22), we ensure that the pre-shared secret $\bar{K}'$ is written into a temporary variable *tmp* (initialized with a random value), instead of T. $\bar{K}'$ is copied into T if ciphertext comparison succeeds ($l = 0$), else z is copied into T upon failure of ciphertext comparison. The check of whether ($l = 0$) is done for every byte copied into T. This simple implementation fix ensures that skipping the conditional move operation does not reveal any information about the pre-shared secret $\bar{K}'$ for invalid ciphertexts, thereby thwarting key recovery. We refer to the aforementioned fixes together as Protect_CT_Compare countermeasure throughout this work.

5.3 Protecting the Message Encoding/Decoding Operation

SCA targeting the message encoding (Encode) and decoding procedures (Decode) are very potent, given that an attacker can perform message recovery in a single trace [3, 42, 51]. Moreover, leakage from these operations can also be utilized as a FD oracle for for key recovery [56]. Ravi *et al.* [42] showed that shuffling countermeasure for the message encoding and decoding operations can be broken in a static-key setting through CCA, in a few hundred to few thousand queries

and Ngo *et al.* [36] also demonstrated break of the combined shuffling and masking scheme in the same manner. Though shuffling does not offer concrete protection, it reasonably increases the attacker's effort to perform message recovery as shown in [36]. Moreover, in an ephemeral setting, shuffling provides concrete protection, as all the aforementioned attacks require a few hundred to few thousand queries for message recovery [42]. Thus, we implement the shuffling countermeasure for the encoding and decoding operations of Kyber KEM. We henceforth refer to this countermeasure as Shuffle_Encode_Decode in this paper.

5.4 Protecting the NTT

SCA targeting the NTT are capable of performing key recovery and message recovery in a single trace [39, 41] or very few traces [27] (NTT_Leakage attacks). Ravi *et al.* [47] proposed a range of generic shuffling and masking countermeasures with varying granularity for the NTT to protect against the aforementioned single trace attacks. Shuffling the order of butterfly operations within the NTT serves as a strong countermeasure against the single trace attacks targeting the NTT operation. They proposed a range of generic shuffling countermeasures for the NTT - which provide a well-defined trade-off between the shuffling entropy (security) and performance. Based on the perceived level of threat from a potential attacker and acceptable performance, the designer can choose the appropriate shuffling countermeasure for the NTT operation.

They also propose randomization of the twiddle factors within the NTT as a potential countermeasure against single trace attacks on the NTT. The basic idea is to *multiplicatively mask* the twiddle constants, such that the twiddle constants within the NTT are randomized. This has the effect of randomizing the internal computations within the NTT, while also ensuring that the attacker cannot build templates for multiplication with known twiddle constants in the butterfly operation. They propose a range of generic masking countermeasures which also establish a well-defined trade-off between security and performance against single trace attacks. For more details, we refer to [47] for the proposed masking and shuffling countermeasure for the NTT. We refer to all the aforementioned countermeasures together as Shuffled_Masked_NTT throughout this paper.

5.5 Protecting the Sampling of Secrets and Errors

Ravi *et al.* [48] demonstrated that nonce-reuse can be induced through faults, in the key-generation and encryption procedure of Kyber KEM, for key recovery and message recovery attacks respectively. Thus, a trivial protection against this attack could be to perform redundant computation of the sampling procedure. While this does not completely offer complete protection, it significantly increases the attacker's complexity. We refer to this as the Redundant_Sample countermeasure in this paper.

Putting it all together, refer Tab.2 for the tabulation of the presented countermeasures against reported SCA and FIA on Kyber KEM. We observe that the presented countermeasures, together can be used to thwart all but one attack (i.e.) SCA_MR_CCA_Protected_Encode_Decode_Leakage attack of Ngo *et al.* [36], which is a message recovery attack done in a CCA setting. Since the attack utilizes invalid ciphertexts, the only concrete countermeasure is to refresh the keys upon observing a certain number of decapsulation failures. While they required $257 \times N$ ($N = 10$) queries for message recovery over Saber KEM, we anticipate that a similar number of queries might be sufficient to perform message recovery over Kyber KEM.

Table 2. Tabulation of the effectiveness of the presented countermeasures against reported SCA and FIA on Kyber KEM. We utilize the following numbers to denote the different countermeasures - (1) Shuffled_Masked_NTT, (2) CT_Sanity_Check, (3) Message_Poly_Sanity_Check (4) Protect_CT_Compare, (5) Shuffle_Encode_Decode, (6) Redundant_Compare

Attack	Countermeasure					
	(1)	(2)	(3)	(4)	(5)	(6)
SCA						
SCA_KR_KCA_NTT_Leakage [41]	✓	✗	✗	✗	✗	✗
SCA_MR_KCA_Encode_Decode_Leakage [3, 42, 51]	✗	✗	✗	✗	✓	✗
SCA_MR_KCA_NTT_Leakage [39]	✓	✗	✗	✗	✗	✗
SCA_KR_CCA_PC_Oracle [18, 49]	✗	✓	✓	✗	✗	✗
SCA_KR_CCA_DF_Oracle [11, 21]	✗	✗	✓	✗	✗	✗
SCA_KR_CCA_FD_Oracle [35, 36, 42, 56]	✗	✓	✓	✗	✗	✗
SCA_KR_CCA_NTT_Leakage [27]	✓	✗	✓	✗	✗	✗
SCA_MR_CCA_Protected_Encode_Decode_Leakage [36]	✗	✗	✗	✗	✗	✗
FIA						
FIA_KR_KCA_Nonce_Reuse [48]	✗	✗	✗	✗	✗	✓
FIA_MR_KCA_Nonce_Reuse [48]	✗	✗	✗	✗	✗	✓
FIA_KR_CCA_Skip_CT_Compare [55]	✗	✓	✓	✓	✗	✗
FIA_KR_CCA_Fault_Assisted_CCA [19, 28, 40]	✗	✗	✓	✗	✗	✗

6 SIDE-CHANNEL AND FAULT-INJECTION ATTACKS ON DILITHIUM

In this section, we present a brief survey of the side-channel and fault injection attacks, that have targeted the Dilithium signature scheme or those attacks, that are applicable to the same. We utilize the algorithm of Dilithium in Alg.3-4 for our analysis.

6.1 Fault Injection Attacks on Dilithium

The signing procedure of Dilithium remains the main target of fault injection attacks, as the signing procedure utilizes the long-term secret key *sk* to generate signatures. Among all the operations in the signing procedure, generation of the primary signature component $\mathbf{z} = \mathbf{s}_1 \cdot \mathbf{c} + \mathbf{y}$ (Line 24 in Alg.3) serves as an attractive target for fault injection attacks. While both $\mathbf{z}$ and $\mathbf{c}$ are revealed as part of the signature, $\mathbf{y}$ is the ephemeral masking polynomial that is used to mask the sensitive intermediate variable ($\mathbf{s}_1 \cdot \mathbf{c}$). We refer to $\mathbf{s}_1$ as the primary secret, since the knowledge of $\mathbf{s}_1$ is sufficient to forge signatures of Dilithium, as shown in [46]. Injection of faults in any of the operations used to generate $\mathbf{z}$, helps the attacker derive a direct relation of the faulty signature $\mathbf{z}^*$ with the primary secret $\mathbf{s}_1$, thereby naturally becoming a target of several reported attacks [12, 16, 22].

6.1.1 Differential Fault Attacks (DFA). The deterministic variant of Dilithium has been target of different style fault attacks, similar to that of deterministic ECC-based signature schemes [2, 8]. The first such attack was proposed by Bruinderink and Pessl [16], whose attack only required a single random fault anywhere over a large window of ~68% of the execution time of the signing procedure, to recover the full secret key. They demonstrated their attack through clock glitching on Deterministic Dilithium running on the ARM Cortex-M4 microcontroller. Refer [16] for more details on the different operations that can be targeted using DFA for key recovery.

We briefly sketch below the main idea of this attack. The attacker lets the target generate a valid signature for a message m. Let the corresponding signature be $\mathbf{z} = \mathbf{s}_1 \cdot \mathbf{c} + \mathbf{y}$. Subsequently, he queries the target to sign the same message m, but now injects a random fault, so as to fault the computation of $\mathbf{c}$ to $\mathbf{c}^*$, but while also ensuring the use of same nonce $\mathbf{y}$ to generate the faulty signature $\mathbf{z}^*$. The difference between the correct and faulty signature yields $\Delta\mathbf{z} = \mathbf{s}_1 \cdot \Delta\mathbf{c}$, which can be easily solved through Gaussian elimination to recover the secret $\mathbf{s}_1$. We refer to this attack using the label Generic_DFA.

Subsequently, Ravi *et al.* [46] presented a practical *skip-addition* fault attack using EMFI on the implementation of Deterministic Dilithium on the ARM Cortex-M4 microcontroller. They proposed to unmask single coefficients of the nonce $\mathbf{y}$ by faulting the final addition operation of single coefficients of $(\mathbf{s}_1 \cdot \mathbf{c})$ with the nonce $\mathbf{y}$ (Line 24). If the attacker faults the addition of the first coefficient of $\mathbf{z}$ (i.e.) $\mathbf{z}[0]$, then $\Delta\mathbf{z}$ yields the first coefficient of $\mathbf{s}_1\mathbf{c}$ (i.e.) $\mathbf{s}_1\mathbf{c}[0]$. With enough faulty signatures (few hundred), the attacker can build a fully solveable linear system of equations, to recover the complete secret $\mathbf{s}_1$. Since the attack relies on simple skipping faults, it is also possible to perform the attack using low cost-FIA attack vectors such as clock-glitching or voltage-glitching. We refer to this attack using the label Skip_Add.

The aforementioned DFA style attacks can only be applied on the deterministic variant, since the attacker requires to induce the same computations in the target signing procedure across multiple executions. However, this is not possible in the probabilistic variant, as the nonce $\mathbf{y} \in R_q^\ell$ is sampled from random in each execution, thereby thwarting differential analysis and therefore key recovery.

6.1.2 Loop Abort Fault Attacks. Espitau *et al.* [22] proposed a novel fault attack to directly target the nonce $\mathbf{y}$ in Fiat-Shamir Abort based signature schemes such as GLP signature scheme [25]. They also practically validated their attack through clock-glitching on the 8-bit Atmel XMEGA128 microcontroller. They propose to use faults to prematurely abort the loop, that samples single coefficients of $\mathbf{y}$ (Line 19 in Alg.3), thereby resulting in generation of nonces with low degrees. In other words, by skipping the loop that samples individual coefficients of $\mathbf{y}$, one can ensure that the remainining cofficients of $\mathbf{y}$ are unsampled, and there is a high chance that these unsampled coefficients have a value of 0. If so, the faulted signature $\mathbf{z}$ contains several coefficients which are nothing but the unmasked coefficients of the product $\mathbf{s}_1 \cdot \mathbf{c}$. The authors show that a single targeted fault in the sampling procedure of $\mathbf{y}$ can result in full key recovery. Though this attack was only demonstrated on the GLP signature scheme [25], this attack can potentially be applied to Dilithium for full key recovery. Since this attack does not involve differential analysis, it is therefore applicable to both the probabilistic and deterministic variants of Dilithium. We refer to this attack using the label Loop_Abort.

6.1.3 Attacks targeting the Verification Procedure. While the aforementioned attacks target the signing procedure, the verification procedure could also serve as a good target for fault injection attacks. One of the main motivation being, forceful acceptance of invalid signatures through faults, for any message of the attacker's choice. However, to the best of our knowledge, we are not aware of a practical fault attack targeting the verification procedure of Dilithium, or any other related lattice-based signature scheme.

One of the obvious targets for fault injection is to simply skip the final comparison operation that decides the validity of the received signatures. In particular, bypassing the comparison of the received challenge polynomial $\mathbf{c}$ with the recomputed challenge polynomial $\bar{\mathbf{c}}$ (Line 6 in Alg.4) ensures successful signature verification. This attack is very similar to the skipping attack demonstrated by Xagawa *et al.* [55], targeting the ciphertext comparison operation in the decapsulation procedure. We refer to this attack using the label Verification_Bypass.

6.2 Side-Channel Attacks on Dilithium

Compared to FIA, there have been relatively much fewer works on SCA over the Dilithium signature scheme. Existing works have mainly targeted the polynomial multiplier used in Dilithium. The first side-channel attack in this direction was proposed by Ravi *et al.* [45], who demonstrated a single-trace horizontal style DPA attack targeting the operation $\mathbf{s}_1 \cdot \mathbf{c}$, implemented using the school-book polynomial multiplier. However, they only demonstrated a simulated attack assuming idealized leakage models, and to some extent, evaluated the effect of leakage noise. Moreover, NTT is the actual polynomial multiplication algorithm used in Dilithium, and thus this attack is no more applicable to the latest implementations of Dilithium. Given that NTT is used for polynomial multiplication, the NTT_Leakage attacks proposed in [39, 41] also become relevant for Dilithium, albeit with appropriate modifications. While proof of leakage from other operations such as the rounding functions (LowBits, HighBits) and rejection sampling has been shown by Migliore *et al.* [32], a full key recovery attack on Dilithium has not been performed, targeting any other operation within the Dilithium signature scheme.

7 PROTECTING DILITHIUM AGAINST SCA/FIA

In this section, we present a range of countermeasures to protect against the different types of SCA/FIA mounted on the Dilithium signature scheme.

7.1 Protection against DFA

A close observation of the faulty signatures generated by the Generic_DFA attacks in [16, 46] reveal that the generated faulty signatures are invalid, which cannot be verified correctly. Thus, verifying the generated signatures serves as a concrete countermeasure against such attacks. While the same countermeasure has been proposed in [16], its overhead on the performance of the signing procedure has not yet been evaluated in any prior work. We refer to this countermeasure as the Verify_After_Sign countermeasure for Dilithium. One of the downsides of this countermeasure is that, it cannot detect faults that are injected directly in the sampling procedure of $\mathbf{y}$ (Line 19), such as the Loop_Abort fault attack of $\mathbf{y}$ [22]. This is because the design of the signature scheme ensures that the authenticity of $\mathbf{y}$ cannot be verified through the signatures. Thus, the sampling of $\mathbf{y}$ is required to be hardened by other dedicated countermeasures.

7.2 Protecting sampling of nonce y

The Loop_Abort fault attack targets the sampling of $\mathbf{y}$, to create low-degree nonces to generate signatures. This requires to perform an early abortion of the loop sampling single coefficients of $\mathbf{y}$. This attack can thus be easily prevented by simply assigning a loop counter to keep track of the number of sampling coefficients of $\mathbf{y}$. This, ensures that an attacker cannot simply skip the sampling of several coefficients. Moreover, a second level of protection can be added by initializing $\mathbf{y}$ with random values, such that skipping of sampling of $\mathbf{y}$ still ensures that $\mathbf{y}$ contains non-zero random values, thereby preventing the attack. A third level of protection can be added by validating the distribution of the coefficients of the sampled $\mathbf{y}$, similar to the CT_Sanity_Check countermeasure proposed for Dilithium. We refer to these countermeasures that protect sampling of $\mathbf{y}$ against fault attacks together as the Protect_YGen countermeasure for Dilithium.

7.3 Protecting the verification procedure

A preliminary analysis of the verification procedure of Dilithium from the *pqm4* library [29] suggests that an attacker can use a single skip fault to skip the verification of the computed challenge polynomial $\mathbf{c}$ with that of the received polynomial (Line 44). Thus, we propose to use the same dynamic loop countermeasure proposed for Kyber KEM (Protect_CT_Compare) to keep track of the number of coefficients of $\mathbf{c}$ that were successfully compared. This helps protect against attacks that work by simply skipping the equality check operation. We refer to this as Protect_Verify_Compare countermeasure for Dilithium.

7.4 Protecting the NTT

Since Dilithium utilizes NTT for polynomial multiplication, the NTT_Leakage attacks mounted on Kyber KEM are also applicable to Dilithium. We implement the shuffling and masking countermeasures proposed by Ravi *et al.* [47] for those NTT instances over the sensitive variables $\mathbf{y}$, $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{t}_0$ in the signing procedure of Dilithium. We refer to all the aforementioned countermeasures together as Shuffled_Masked_NTT throughout this report.

Putting it all together, refer Tab.3 for the tabulation of the presented countermeasures against reported SCA and FIA on Dilithium. We observe that the presented countermeasures, together can be used to thwart all the reported attacks on Dilithium.

Table 3. Tabulation of the effectiveness of the presented countermeasures against reported SCA and FIA on Dilithium signature scheme. We utilize the following numbers to denote the different countermeasures - (1) Shuffled_Masked_NTT, (2) Verify_After_Sign, (3) Verify_YGen (4) Protect_Verify_Compare

Attack	**Countermeasure**			
	(1)	(2)	(3)	(4)
SCA				
NTT_Leakage	✓	✗	✗	✗
FIA				
Generic_DFA	✗	✓	✗	✗
Skip_Add	✗	✓	✗	✗
Loop_Abort	✗	✗	✓	✗
Verification_Bypass	✗	✗	✗	✓

8 EXPERIMENTAL EVALUATION

In this section, we perform a practical performance evaluation of the presented countermeasures when integrated into optimized software implementations of Kyber and Dilithium, running on the following two embedded platforms - (1) STM32F4 microcontroller based on the ARM Cortex-M4 processor (2) Raspberry Pi 3 Model B Plus based on the ARM Cortex-A53 processor. While the STM32F4 microcontroller is a representative for an low-power embedded microcontroller, the Raspberry Pi 3 device is a representative of a more sophisticated handheld gadget.

8.1 Target Platform and Implementation Details

8.1.1 ARM Cortex-M4 based Platform. Our target platform for the ARM Cortex-M4 processor is the STM32F4DISCOVERY board (DUT) housing the STM32F407 microcontroller and the clock frequency is 24 MHz. Our countermeasures were implemented on the M4-optimized implementations of Kyber and Dilithium available in the public *pqm4* library [29], a benchmarking framework for PQC schemes on the ARM Cortex-M4 microcontroller. The M4-optimized implementation of Kyber is based on the memory efficient high-speed implementation proposed by Botros, Kannwischer and Schwabe in [15]. The M4-optimized implementation is based on compact Dilithium optimizations reported by Greconici, Kannwischer and Sprenkels in [24]. [1] Their work builds upon the early evaluation optimization by Ravi et al. [43] and additionally proposes faster asssembly implementations of NTT for the Cortex-M4. All implementations were compiled with the arm-none-eabi-gcc-7.3.1 compiler using compiler flags -O3 -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16.

8.1.2 ARM Cortex-A53 based Platform. Our target platform for the ARM Cortex-A53 processor is the Raspberry Pi 3 Model B Plus Rev 1.3, running Debian GNU/Linux 11 at 1.4 GHz. We perform evaluation of countermeasures on the NEON-optimized implementations of Kyber and Dilithium available in the *liboqs* library [52], an open source C library for quantum-safe cryptographic algorithms. The NEON optimized implementations of Kyber and Dilithium is based on the work of Becker *et al.* [9]. All implementations were compiled with the aarch64-none-linux-gnu-10.3-2021.07 compiler at the highest optimization level -O3.

We have implemented the countermeasures on both Kyber and Dilithium such that, the required countermeasures can be independently turned on/off based on the designer's security requirements.

8.2 Experimental Results for Kyber KEM

Refer Tab.4 and Tab.6 for the performance overheads due to the Shuffled_Masked_NTT countermeasures against the NTT_Leakage attacks on Kyber KEM, running on the ARM Cortex-M4 and ARM Cortex-A53 devices respectively. While we have implemented all the shuffling (3) and masking (4) countermeasures proposed in [47], for brevity, we only report numbers for the countermeasures referred to as Coarse_Shuffled_NTT and Generic_2_Masked_NTT (Refer to [47] for the terminology used for the different Shuffled_Masked_NTT countermeasures).

On the ARM Cortex-M4 device, we observe a performance impact in the range of 45 – 71% for key generation, 44-75% for the encapsulation and 53 – 99% for the decapsulation procedure, across all parameters of Kyber KEM. However, on the ARM Cortex-A53 device, we observe a much higher performance impact between 163-219% for key generation, 189-258% for encapsulation and 226-352% for decapsulation. We note that the optimized NTT implementations on the target devices are implemented in pure assembly (M4-optimized and NEON optimized), but the countermeasures are implemented over the C-based NTT/INTT implementations, resulting in high performance overheads, particularly on the ARM Cortex-A53 device. Thus, we argue that it is possible to obtain significantly improved overheads, provided that the protected NTT/INTTs are implemented in assembly.

[1] Our analysis and experiments were carried out on the implementations of Kyber and Dilithium corresponding to the commit hash `2691b4915b76db8b765ba89e4e09adc6b999763f`, and were available in the *pqm4* library until Jan 31, 2022. However, our attacks also apply in the same manner to the most recent NTT implementations in the *pqm4* library.

Table 4. Performance Comparison of the Shuffled_Masked_NTT countermeasures for Kyber KEM, compared to the optimized unprotected implementations on the ARM Cortex-M4 device. Numbers were obtained on the STM32F407VG microcontroller mounted on the STM32F407DISCOVERY board, running at 24 MHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^3$)**								
	KeyGen			**Encaps**			**Decaps**		
	Unprot.	**Prot.**	**Ovh. (%)**	**Unprot.**	**Prot.**	**Ovh. (%)**	**Unprot.**	**Prot.**	**Ovh. (%)**
Coarse_Shuffled_NTT									
Kyber512	457.0	782.9	71.3	552.1	971.1	75.9	511.5	1017.8	99
Kyber768	748.6	1238.0	65.4	903.6	1486.0	64.4	842.4	1506.7	78.8
Kyber1024	1188.2	1840.9	54.9	1378.6	2124.2	54.1	1298.0	2125.5	63.8
Generic_2_Masked_NTT									
Kyber512	457.0	728.5	59.4	552.1	898.9	62.8	511.5	933.5	82.5
Kyber768	748.6	1155.5	54.4	903.6	1385.9	53.4	842.4	1399.9	66.2
Kyber1024	1188.2	1731.5	45.7	1378.6	1997.1	44.9	1298.0	1991.7	53.4

Table 5. Performance Comparison of the custom SCA-FIA countermeasures for Kyber KEM, compared to the optimized unprotected implementation on the ARM Cortex-M4 device. Numbers were obtained on the STM32F407VG microcontroller mounted on the STM32F407DISCOVERY board, running at 24 MHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^3$)**		
	Decaps		
	Unprot.	**Prot.**	**Ovh. (%)**
CT_Sanity_Check			
Kyber512	511.5	690.0	34.9
Kyber768	842.4	1028.6	22.1
Kyber1024	1298.0	1492.3	15
Message_Poly_Sanity_Check			
Kyber512	511.5	675.6	32.1
Kyber768	842.4	1006.1	19.4
Kyber1024	1298.0	1474.7	13.6
Protect_CT_Compare			
Kyber512	511.5	579.9	13.4
Kyber768	842.4	910.1	8
Kyber1024	1298.0	1365.8	5.2
Shuffle_Encode_Decode			
Kyber512	511.5	520.6	1.8
Kyber768	842.4	854.2	1.4
Kyber1024	1298.0	1311.2	1

It is also important to note that the NTT_Leakage attacks have only been demonstrated on the ARM Cortex-M4 device, in a profiled setting, assuming attacker's in-depth knowledge of the target device as well as implementation.

Thus, these attacks are not trivial to be mounted on more complex platforms such as the ARM Cortex-A53 device, which run an embedded linux distribution at a much higher frequency (1.4 GHz), compared to the ARM Cortex-M4 (168 MHz). Thus, in a practical setting, Shuffled_Masked_NTT countermeasures for the ARM Cortex-A53 device might not be necessary and could serve as an overkill.

Table 6. Performance Comparison of the Shuffled_Masked_NTT countermeasures for Kyber KEM, compared to the optimized unprotected implementations on the ARM Cortex-A53 device. Numbers were obtained on the Raspberry Pi 3 Model B Plus Rev 1.3 based on the ARM Cortex-A53 processor, running Debian GNU/Linux 11 at 1.4 GHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	Clock Cycles ($\times 10^3$)								
	KeyGen			Encaps			Decaps		
	Unprot.	Prot.	Ovh. (%)	Unprot.	Prot.	Ovh. (%)	Unprot.	Prot.	Ovh. (%)
Coarse_Shuffled_NTT									
Kyber512	108.3	345.7	219	128.7	461.0	258.2	114.5	518.6	352.7
Kyber768	162.5	519.3	219.5	193.1	658.8	241	177.4	714.7	302.9
Kyber1024	244.3	719.1	194.3	285.4	880.1	208.4	267.6	934.9	249.3
Generic_2_Masked_NTT									
Kyber512	108.3	308.0	184.2	128.7	429.9	234	114.5	484.2	322.8
Kyber768	162.5	462.1	184.4	193.1	615.6	218.7	177.4	667.8	276.4
Kyber1024	244.3	643.2	163.2	285.4	825.1	189.1	267.6	874.4	226.7

Refer Tab.5 and 7 for the performance overheads due to the CT_Sanity_Check, Message_Poly_Sanity_Check and Shuffle_Encode_Decode countermeasures for Kyber KEM, implemented on the ARM Cortex-M4 and ARM Cortex-A53 devices respectively[2]. On the ARM Cortex-M4 device, these countermeasures impose very reasonable overheads in the range of 15-33%, 13-32%, 5-13% and 1-2% for the different parameter sets of Kyber KEM. On the ARM Cortex-A53 device, the overheads are in the range of $3-4\%$, $\approx 0\%$, $7-18\%$ and $19-39\%$ for Kyber KEM.

8.3 Experimental Results for Dilithium

Refer Tab.8 and Tab.9 for the performance overheads due to the Shuffled_Masked_NTT countermeasures against the NTT_Leakage attacks on Dilithium, implemented on the ARM Cortex-M4 and ARM Cortex-A53 devices respectively. While we have implemented all the shuffling (3) and masking (4) countermeasures proposed in [47], for brevity, we only report numbers for the countermeasures referred to as Coarse_Shuffled_NTT and Generic_2_Masked_NTT.

On the ARM Cortex-M4 device, we observe a performance impact in the range of $22-31\%$ for key-generation and $112-152\%$ for the signing procedure. However, on the ARM Cortex-A53 device, we observe a much higher performance impact between 77-109% for key generation, 438-384% for the signing procedure. We note that the optimized NTT implementations on the target devices are implemented in pure assembly (M4-optimized and NEON optimized), but the countermeasures are implemented over the C-based NTT/INTT implementations, resulting in high performance

[2]We do not report results of the Redundant_Sample countermeasure in the current version, but can report the results in a future version of the paper.

Table 7. Performance Comparison of the custom SCA-FIA countermeasures for Kyber KEM, compared to the optimized unprotected implementations on the ARM Cortex-A53 device. Numbers were obtained on the Raspberry Pi 3 Model B Plus Rev 1.3 based on the ARM Cortex-A53 processor, running Debian GNU/Linux 11 at 1.4 GHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^3$)**		
	Decaps		
	Unprot.	**Prot.**	**Ovh. (%)**
CT_Sanity_Check			
Kyber512	114.5	120.0	4.8
Kyber768	177.4	189.0	6.6
Kyber1024	267.6	277.7	3.8
Message_Poly_Sanity_Check			
Kyber512	114.5	114.1	-0.3
Kyber768	177.4	178.8	0.8
Kyber1024	267.6	268.0	0.2
Protect_CT_Compare			
Kyber512	114.5	135.3	18.1
Kyber768	177.4	200.9	13.3
Kyber1024	267.6	288.3	7.7
Shuffle_Encode_Decode			
Kyber512	114.5	159.7	39.5
Kyber768	177.4	227.5	28.2
Kyber1024	267.6	320.8	19.9

overheads, particularly on the ARM Cortex-A53 device. Thus, we argue that it is possible to obtain significantly improved overheads, provided that the protected NTT/INTTs are implemented in assembly.

Table 8. Performance Comparison of the Shuffled_Masked_NTT countermeasures for Dilithium, compared to the optimized unprotected implementation on the ARM Cortex-M4 device. Numbers were obtained on the STM32F407VG microcontroller mounted on the STM32F407DISCOVERY board, running at 24 MHz. Numbers are provided in terms of $\times 10^6$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^6$)**					
	KeyGen			**Sign**		
	Unprot.	**Prot.**	**Ovh. (%)**	**Unprot.**	**Prot.**	**Ovh. (%)**
Coarse_Shuffled_NTT						
Dilithium2	1.6	2.1	31.9	4.1	9.5	132.1
Dilithium3	2.8	3.5	24.7	6.8	14.4	112.6
Generic_2_Masked_NTT						
Dilithium2	1.6	2.0	30.0	4.1	10.4	152.9
Dilithium3	2.8	3.4	22.5	6.7	16.4	142.0

Table 9. Performance Comparison of the different SCA-FIA countermeasures for Dilithium and the overheads they incur on on optimized implementations on the ARM Cortex-M4 device. Numbers were obtained on the STM32F407VG microcontroller mounted on the STM32F407DISCOVERY board, running at 24 MHz. Numbers are provided in terms of $\times 10^6$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^6$)**		
	Unprot.	**Prot.**	**Ovh. (%)**
Verify_After_Sign (Sign)			
Dilithium2	4.1	4.4	7.7
Dilithium3	6.8	7.2	6.2
Protect_YGen (Sign)			
Dilithium2	4.1	5.6	35.5
Dilithium3	6.8	8.5	25.6
Protect_Verify_Compare (Verify)			
Dilithium2	1.6	1.6	≈ 0
Dilithium3	2.7	2.7	≈ 0

Table 10. Performance Comparison of the Shuffled_Masked_NTT countermeasures for Dilithium, compared to the optimized unprotected implementation on the ARM Cortex-A53 device. Numbers were obtained on the Raspberry Pi 3 Model B Plus Rev 1.3 based on the ARM Cortex-A53 processor, running Debian GNU/Linux 11 at 1.4 GHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^3$)**					
	KeyGen			**Sign**		
	Unprot.	**Prot.**	**Ovh. (%)**	**Unprot.**	**Prot.**	**Ovh. (%)**
Coarse_Shuffled_NTT						
Dilithium2	434.1	908.5	109.3	1329.6	5956.4	348
Dilithium3	745.1	1423.2	91	1987.5	9194.8	362.6
Dilithium5	1205.2	2157.4	79	2527.8	11099.1	339.1
Generic_2_Masked_NTT						
Dilithium2	434.0	894.9	106.2	1329.6	5993.8	350.8
Dilithium3	745.1	1415.5	90	1987.5	9628.5	384.5
Dilithium5	1205.2	2138.9	77.5	2527.8	11331.9	348.3

Refer Tab.10 and 11 for the performance overheads due to the Verify_After_Sign, Verify_YGen and Protect_Verify_Compare countermeasures for Dilithium, implemented on the ARM Cortex-M4 and ARM Cortex-A53 devices respectively. On the ARM Cortex-M4 device, these countermeasures impose very reasonable overheads in the range of 6-7%, 25-35%, and $\approx 0\%$ for the different parameter sets of Kyber KEM. On the ARM Cortex-A53 device, the overheads are much smaller in the range of $15-24\%$, $32-33\%$ and $3-6\%$ for Dilithium.

Table 11. Performance Comparison of the different SCA-FIA countermeasures for Dilithium and the overheads they incur on optimized implementations on the ARM Cortex-A53 device. Numbers were obtained on the Raspberry Pi 3 Model B Plus Rev 1.3 based on the ARM Cortex-A53 processor, running Debian GNU/Linux 11 at 1.4 GHz. Numbers are provided in terms of $\times 10^3$ clock cycles. Ovh denotes overhead in percentage.

Scheme	**Clock Cycles ($\times 10^3$)**		
	Unprot.	**Prot.**	**Ovh. (%)**
Verify_After_Sign (Sign)			
Dilithium2	1329.6	1537.0	15.6
Dilithium3	1987.5	2413.1	21.4
Dilithium5	2527.8	3155.1	24.8
Protect_YGen (Sign)			
Dilithium2	1329.6	1764.0	32.7
Dilithium3	1987.5	2697.8	35.7
Dilithium5	2527.8	3368.3	33.2
Protect_Verify_Compare (Verify)			
Dilithium2	438.6	452.8	3.2
Dilithium3	699.5	740.8	5.9
Dilithium5	1188.3	1259.4	6

Thus, we can observe that barring the Shuffled_Masked_NTT countermeasure, the other countermeasure impose very reasonable overheads on the performance of Kyber and Dilithium, on both the evaluated embedded platforms.

9 CONCLUSION

In this work, we present a systematic study of Side-Channel Attacks (SCA) and Fault Injection Attacks (FIA) on structured lattice-based schemes, with main focus on Kyber and Dilithium. Given the inadequacy of generic countermeasues such as masking to protect against the wide-variety of known attacks [11, 35], we also present a range of custom countermeasures, to protect against the known SCA and FIA reported on both Kyber and Dilithium. This includes two novel countermeasures, to protect the decapsulation procedure against SCA and FIA assisted chosen-ciphertext attacks. Finally, we implement all the presented countermeasures for Kyber and Dilithium, within two well-known public software libraries for PQC - (1) *pqm4* library for the ARM Cortex-M4 based microcontroller and (2) *liboqs* library for the Raspberry Pi 3 Model B Plus based on the ARM Cortex-A53 processor. Our performance evaluation reveals that the presented custom countermeasures incur reasonable performance overheads, on both the evaluated embedded platforms. We therefore believe our work argues for usage of custom countermeasures within real-world implementations of lattice-based schemes, either in a standalone manner, or as reinforcements to generic countermeasures such as masking.

REFERENCES

[1] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. 2020. Status report on the second round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST* (2020).

[2] Christopher Ambrose, Joppe W Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. 2018. Differential attacks on deterministic signatures. In *Cryptographers' Track at the RSA Conference*. Springer, 339–353.

[3] Dorian Amiet, Andreas Curiger, Lukas Leuenberger, and Paul Zbinden. 2020. Defeating NewHope with a single trace. In *International Conference on Post-Quantum Cryptography*. Springer, 189–205.

[4] Daniel Apon and James Howe. 2021. Attacks on NIST PQC 3rd Round Candidates. Invited talk at Real World Crypto 2021, https://iacr.org/submit/files/slides/2021/rwc/rwc2021/22/slides.pdf.

[5] Roberto Avanzi, Joppe W. Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2021. CRYSTALS-Kyber (version 3.02): Algorithm specifications and supporting documentation (Auguest 4, 2021). https://pq-crystals.org/kyber/data/kyber-specification-round3.pdf. (2021).

[6] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. 2018. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 81–88.

[7] Florian Bache, Clara Paglialonga, Tobias Oder, Tobias Schneider, and Tim Güneysu. 2020. High-speed masking for polynomial comparison in lattice-based kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 483–507.

[8] Alessandro Barenghi and Gerardo Pelosi. 2016. A note on fault attacks against deterministic signature schemes (short paper). In *International Workshop on Security*. Springer, 182–192.

[9] Hanno Becker, Vincent Hwang, Matthias J Kannwischer, Bo-Yin Yang, and Shang-Yi Yang. 2022. Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), 221–244.

[10] Michiel Van Beirendonck, Jan-Pieter D'anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. 2021. A side-channel-resistant implementation of SABER. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17, 2 (2021), 1–26.

[11] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel van Beirendonck. 2021. Attacking and Defending Masked Polynomial Comparison for Lattice-Based Cryptography. 2021, 3 (2021), 334–359. https://doi.org/10.46586/tches.v2021.i3.334-359

[12] Nina Bindel, Johannes Buchmann, and Juliane Krämer. 2016. Lattice-based signature schemes and their sensitivity to fault attacks. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on*. IEEE, 63–77.

[13] Joppe W Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. 2018. Assessing the Feasibility of Single Trace Power Analysis of Frodo. *IACR Cryptology ePrint Archive* (2018).

[14] Joppe W Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. 2021. Masking kyber: First-and higher-order implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 173–214.

[15] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. 2019. Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4. In *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings* (2019). 209–228. https://doi.org/10.1007/978-3-030-23696-0_11

[16] Leon Groot Bruinderink and Peter Pessl. 2018. Differential Fault Attacks on Deterministic Lattice Signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018). https://eprint.iacr.org/2018/355.pdf.

[17] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 2020. SABER: Mod-LWR based KEM (Round 3 Submission). https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf. *Submission to the NIST post-quantum project* (2020).

[18] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. Timing attacks on error correcting codes in post-quantum schemes. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. 2–9.

[19] Jeroen Delvaux. 2021. Roulette: Breaking Kyber with Diverse Fault Injection Setups. *Cryptology ePrint Archive* (2021), 1622.

[20] Léo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals–dilithium: Digital signatures from module lattices. https://pq-crystals.org/dilithium/data/dilithium-specification-round3.pdf. *Submission to the NIST's post-quantum cryptography standardization process* (2018).

[21] Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. 2022. Higher-Order Masked Ciphertext Comparison for Lattice-Based Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022, 2 (Feb. 2022), 115–139. https://doi.org/10.46586/tches.v2022.i2.115-139

[22] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. 2016. Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures. In *International Conference on Selected Areas in Cryptography*. Springer, 140–158.

[23] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*. Springer, 537–554.

[24] Denisa OC Greconici, Matthias J Kannwischer, and Daan Sprenkels. 2021. Compact dilithium implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 1–24.

[25] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2012. Practical lattice-based cryptography: A signature scheme for embedded systems. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 530–547.

[26] Qian Guo, Thomas Johansson, and Alexander Nilsson. 2020. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In *Annual International Cryptology Conference*. Springer, 359–386.

[27] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. 2021. Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 88–113.

[28] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. 2021. Fault-enabled chosen-ciphertext attacks on kyber. In *International Conference on Cryptology in India*. Springer, 311–334.

[29] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[30] Vadim Lyubashevsky. 2009. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 598–616.

[31] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* 60, 6 (2013), 43.

[32] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. 2019. Masking dilithium. In *International Conference on Applied Cryptography and Network Security*. Springer, 344–362.

[33] Catinca Mujdei, Arthur Beckers, Jose Bermundo, Angshuman Karmakar, Lennert Wouters, and Ingrid Verbauwhede. 2022. Side-Channel Analysis of Lattice-Based Post-Quantum Cryptography: Exploiting Polynomial Multiplication. *Cryptology ePrint Archive* (2022).

[34] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. 2019. Post-quantum lattice-based cryptography implementations: A survey. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–41.

[35] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. 2021. A side-channel attack on a masked IND-CCA secure Saber KEM implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 676–707.

[36] Kalle Ngo, Elena Dubrova, and Thomas Johansson. 2021. Breaking Masked and Shuffled CCA Secure Saber KEM by Power Analysis. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*. 51–61.

[37] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. 2018. Practical CCA2-Secure and Masked Ring-LWE Implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 142–174.

[38] Judea Pearl. 1986. Fusion, propagation, and structuring in belief networks. *Artificial intelligence* 29, 3 (1986), 241–288.

[39] Peter Pessl and Robert Primas. 2019. More practical single-trace attacks on the number theoretic transform. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 130–149.

[40] Peter Pessl and Lukas Prokop. 2021. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 37–60.

[41] Robert Primas, Peter Pessl, and Stefan Mangard. 2017. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 513–533.

[42] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. 2021. On Exploiting Message Leakage in (few) NIST PQC Candidates for Practical Message Recovery Attacks. *IEEE Transactions on Information Forensics and Security* (2021).

[43] Prasanna Ravi, Sourav Sen Gupta, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Improving speed of Dilithium's signing procedure. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 57–73.

[44] Prasanna Ravi, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2021. Lattice-based key-sharing schemes: A survey. *ACM Computing Surveys (CSUR)* 54, 1 (2021), 1–39.

[45] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2018. Side-channel assisted existential forgery attack on Dilithium-a NIST PQC candidate. *Cryptology ePrint Archive* (2018).

[46] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of NIST candidates. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 427–440.

[47] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. 2020. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 123–146.

[48] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2019. Number "Not Used" Once-Practical Fault Attack on pqm4 Implementations of NIST Candidates. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 232–250.

[49] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. 2020. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 3 (2020), 307–335.

[50] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.

[51] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. 2020. Single-Trace Attacks on Message Encoding in Lattice-Based KEMs. 8 (2020), 183175–183191.

[52] Douglas Stebila and Michele Mosca. 2016. Post-quantum key exchange for the internet and the open quantum safe project. In *International Conference on Selected Areas in Cryptography*. Springer, 14–37.

[53] Felipe Valencia, Tobias Oder, Tim Güneysu, and Francesco Regazzoni. 2018. Exploring the Vulnerability of R-LWE Encryption to Fault Attacks. In *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*. ACM.

[54] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. 2014. Soft analytical side-channel attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 282–296.

[55] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. 2021. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 33–61.

[56] Zhuang Xu, Owen Michael Pemberton, Sujoy Sinha Roy, David Oswald, Wang Yao, and Zhiming Zheng. 2021. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Trans. Comput.* (2021).