Roulette: Breaking Kyber with Diverse Fault Injection Setups

Jeroen Delvaux and Santos Merino Del Pozo

Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE
jeroen.delvaux@tii.ae, santos@tii.ae

Abstract. At Indocrypt 2021, Hermelink, Pessl, and Pöppelmann presented a fault injection attack against KYBER's decapsulation module. The attack can thwart countermeasures such as masking, shuffling, and double executions, but is not overly easy to perform. In this work, we extend and facilitate the attack in two ways, thereby admitting a larger variety of fault injection setups. Firstly, the attack surface is enlarged: originally, the two input operands of the polynomial comparison are covered, and we additionally cover encryption modules such as binomial sampling, butterflies in the last layer of the inverse *number-theoretic transform* (NTT), modular reduction, and ciphertext compression. Secondly, the fault model is relaxed: originally, precise bit flips are required, and we additionally support set-to-0 faults, set-to-1 faults, random faults, arbitrary bit flips, instruction skips, etc. A notable feature of our attack is that masking and certain forms of blinding help the attack. If finite field elements are visualized in a circular manner, our attack is analogous to the casino game *roulette*: randomization-based countermeasures spin the wheel, and the attacker only needs to wait for a certain set of pockets.

Keywords: Fault Attack · Kyber · Key-Encapsulation Mechanism · Lattice-Based Cryptography · Post-Quantum Cryptography

1 Introduction

KYBER [ABD+20] is a lattice-based *key-encapsulation mechanism* (KEM) and, at the time of writing this paper, one of the round 3 finalists in the on-going *post-quantum cryptography* (PQC) standardization process run by the United States' *National Institute of Standards and Technology* (NIST). Although the selection process initially focused on the mathematical strength and the implementation efficiency of the proposals, the resistance to *side-channel analysis* (SCA) and *fault injection* (FI) attacks became a major topic towards the end—i.e., at round 3. In this work, we revisit the FI attack proposed by Hermelink et al. [HPP21] at Indocrypt 2021, which requires a precise bit flip (e.g., induced with a laser beam) at either input operand of the polynomial comparison.

1.1 Contributions of this paper

Although the attack of Hermelink et al. [HPP21] is efficient, the time and expertise needed to prepare and calibrate such a specialized FI setup is substantial—but unaccounted for. Especially in scenarios where a single device instead of a batch of devices is targeted, the time spent on building the setup has no advantages of scale and likely surpasses the time needed for the actual key-recovery. Motivated by this disparity, we make the attack of Hermelink et al. [HPP21] accessible to a large variety of possibly low-budget adversaries—by relying on an avalanche of faulty intermediates, it happens that almost any fault is a good fault. We also extend the attack surface such that these arbitrary faults can be injected into previously untargeted computations. The polynomial comparison was already identified as a prime target for FI attacks [OSPG18, HPP21, XIU+21], and we forewarn secure-system designers that an additional set of building blocks, including binomial sampling, butterflies in the last layer of the inverse *number-theoretic transform* (NTT), modular reduction, and ciphertext compression, should be protected against FI attacks. These new targets also enable an attacker to bypass a redundancy countermeasure proposed by Hermelink et al. [HPP21]. Finally, our manuscript lays out a peculiar case where masking and certain forms of blinding facilitate an FI attack—normally, these SCA countermeasures either decrease the vulnerability to FI attacks or result in a *status quo*. Due to the inherent hunger for true randomness in prime-field elements, which are often represented by a circle, our attack methodology is nicknamed *Roulette*.

1.2 Organization of this paper

Section 2 reviews the specifications of Kyber, which is the KEM targeted by our attacks. Sections 3 and 4 consist of background and related work on SCA and FI against lattice-based KEM implementations, respectively. Section 5 presents the Roulette methodology, and its application to Kyber's decapsulation. Finally, we conclude our paper in Section 6.

1.3 Notation

Variables and constants are denoted by characters from the Latin and Greek alphabets, respectively. Vectors and matrices of polynomials are severally denoted by bold lowercase and bold uppercase characters. Functions are printed in a sans-serif font, e.g., F. With $\lceil\cdot\rfloor$, we denote rounding to the nearest integer where ties (fraction of exactly 0.5) are rounded up.

2 Kyber

As for other lattice-based KEMs, Kyber [ABD+20] starts from a *public-key encryption* (PKE) scheme that is secure against *chosen-plaintext attacks* (CPAs), as recapitulated in Section 2.1, and to which a variation of the *Fujisaki–Okamoto* (FO) transform is applied to additionally resist *chosen-ciphertext attacks* (CCAs), as summarized in Section 2.2. We abstain from comprehensive descriptions and only highlight aspects that are important for this work.

2.1 Public-Key Encryption

The PKE scheme consists of key generation, encryption, and decryption, as specified in simplified form in Algorithms 1 to 3 respectively. The security of the scheme is based on the *module learning with errors* (MLWE) problem. Polynomial arithmetic is performed in the ring $\mathbb{R}_{(\rho,\eta)} = \mathbb{Z}_\rho[x]/(x^\eta + 1)$, where degree $\eta = 256$ of the irreducible polynomial is a power of two and where prime $\rho = 3329 = 256 \cdot 13 + 1$ so that the η-th root of unity exists, i.e., $\zeta^{256} \mod \rho = 1$ where $\zeta = 17$. These design choices allow polynomial multiplications to be realized with quasilinear time complexity $\mathcal{O}(\eta \cdot \log_2 \eta)$ through the *number-theoretic transform* (NTT) according to Equation (1), where operator $\circ$ comprises $\eta/2 = 128$ products of linear polynomials.

$$a[x] \cdot b[x] \mod (x^\eta + 1) = \mathsf{INTT}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b)). \tag{1}$$

The NTT and the *inverse NTT* (INTT) both consist of $\log_2(\eta) - 1 = 7$ layers that each contains $\eta/2 = 128$ butterfly operations $\mathsf{Butterfly} : \mathbb{Z}_\rho^2 \to \mathbb{Z}_\rho^2$. The INTT is typically

Algorithm 1 Kyber.CPAPKE.KeyGen(): key generation

Output: Public key p
Output: Private key $\hat{\mathbf{s}}$
1: $d \leftarrow \{0,1\}^{256}$
2: $(q, b) \leftarrow \mathsf{G}(d)$
3: $\hat{\mathbf{A}} \leftarrow \mathsf{Parse}(\mathsf{XOF}(q))$ ▷ Generate uniform matrix $\hat{\mathbf{A}}$ in the NTT domain
4: $(\mathbf{s}, \mathbf{e}) \leftarrow \mathsf{CBD}(\mathsf{PRF}(b))$ ▷ Sample from a *centered binomial distribution* (CBD)
5: $\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$
6: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \mathsf{NTT}(\mathbf{e})$ ▷ $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$
7: $p \leftarrow \hat{\mathbf{t}} \| q$

Algorithm 2 Kyber.CPAPKE.Enc(p, m, r): encryption

Input: Public key $p \triangleq \hat{\mathbf{t}} \| q$
Input: Message m
Input: Coins r
Output: Ciphertext $c \triangleq (\mathbf{u}, v)$
1: $\hat{\mathbf{A}} \leftarrow \mathsf{Parse}(\mathsf{XOF}(q))$ ▷ Regenerate uniform matrix $\hat{\mathbf{A}}$
2: $(\mathbf{r}, \mathbf{e}_1, e_2) \leftarrow \mathsf{CBD}(\mathsf{PRF}(r))$ ▷ Sample noise from binomial distribution
3: $\hat{\mathbf{r}} \leftarrow \mathsf{NTT}(\mathbf{r})$
4: $\mathbf{u} \leftarrow \mathsf{INTT}(\hat{\mathbf{A}}^{\mathsf{T}} \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{u} \leftarrow \mathbf{A}^{\mathsf{T}}\mathbf{r} + \mathbf{e}_1$
5: $m \leftarrow \mathsf{Decompress}(m; \rho, 1)$ ▷ Error-free message; see Eq. (4)
6: $v \leftarrow \mathsf{INTT}(\hat{\mathbf{t}}^{\mathsf{T}} \circ \hat{\mathbf{r}}) + e_2 + m$ ▷ $v \leftarrow \mathbf{t}^{\mathsf{T}}\mathbf{r} + e_2 + m$
7: $\mathbf{u} \leftarrow \mathsf{Compress}(\mathbf{u}; \rho, \delta_u)$ ▷ See Eq. (3)
8: $v \leftarrow \mathsf{Compress}(v; \rho, \delta_v)$ ▷ See Eq. (3)

Algorithm 3 Kyber.CPAPKE.Dec$(\hat{\mathbf{s}}, c)$: decryption

Input: Private key $\hat{\mathbf{s}}$
Input: Ciphertext $c \triangleq (\mathbf{u}, v)$
Output: Message m
1: $\mathbf{u} \leftarrow \mathsf{Decompress}(\mathbf{u}; \rho, \delta_u)$ ▷ See Eq. (4)
2: $v \leftarrow \mathsf{Decompress}(v; \rho, \delta_v)$ ▷ See Eq. (4)
3: $\hat{\mathbf{u}} \leftarrow \mathsf{NTT}(\mathbf{u})$
4: $m \leftarrow v - \mathsf{INTT}(\hat{\mathbf{s}}^{\mathsf{T}} \circ \hat{\mathbf{u}})$ ▷ $m \leftarrow v - \mathbf{s}^{\mathsf{T}}\mathbf{u}$
5: $m \leftarrow \mathsf{Compress}(m; \rho, 1)$ ▷ LWE error correction according to Eq. (5)

implemented using the *Gentleman–Sande* (GS) butterfly in Equation (2), where twiddle factor τ is a power of the root of unity ζ.

$$\mathsf{GSButterfly}(a, b; \tau) \triangleq \big(a + b, (a - b)\,\tau\big) \mod \rho. \tag{2}$$

The lossy compression is defined in Equations (3) and (4). For Kyber512 and Kyber768, the compression of ciphertext $(\mathbf{u}, v)$ uses constants $(\delta_u, \delta_v) = (10, 4)$, whereas for Kyber1024, $(\delta_u, \delta_v) = (11, 5)$. The compression of message coefficients $m \in \mathbb{Z}_\rho$ in Line 5 of Algorithm 3 uses $\delta = 1$, which implies that Equation (3) boils down to Equation (5).

$$\mathsf{Compress}(x; \rho, \delta) = \lceil 2^d x/\rho \rfloor \mod 2^\delta. \tag{3}$$

$$\mathsf{Decompress}(x; \rho, \delta) = \lceil \rho\, x/2^\delta \rfloor. \tag{4}$$

$$\mathsf{Compress}(x; \rho, 1) = \begin{cases} 1 & \text{if } q/4 < x < 3q/4, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Algorithm 4 $\mathsf{Kyber.CCAKEM.KeyGen}()$: key generation

Output: Public key p
Output: Private key s

$z \leftarrow \{0,1\}^{256}$
$(p, \hat{\mathbf{s}}) \leftarrow \textsc{Kyber}.\mathsf{CPAPKE.KeyGen}()$
$h \leftarrow \mathsf{H}(p)$
$s \leftarrow \hat{\mathbf{s}} \| p \| h \| z$

Algorithm 5 $\mathsf{Kyber.CCAKEM.Enc}(p)$: key encapsulation

Input: Public key p
Output: Ciphertext c
Output: Symmetric key k

$m \leftarrow \{0,1\}^{256}$
$m \leftarrow \mathsf{H}(m)$
$(k, r) \leftarrow \mathsf{G}(m \| \mathsf{H}(p))$
$c \leftarrow \textsc{Kyber}.\mathsf{CPAPKE.Enc}(p, m, r)$ ▷ Encryption with message-derived seed
$k \leftarrow \mathsf{KDF}(k \| \mathsf{H}(c))$

Algorithm 6 $\mathsf{Kyber.CCAKEM.Dec}(c, s')$: key decapsulation

Input: Ciphertext c
Input: Private key $s' \triangleq \hat{\mathbf{s}} \| p \| h \| z$
Output: Symmetric key k

$m \leftarrow \textsc{Kyber}.\mathsf{CPAPKE.Dec}(\hat{\mathbf{s}}, c)$ ▷ CPA-secure decryption with private key
$(k, r) \leftarrow \mathsf{G}(m \| h)$ ▷ Regenerate seed for encryption
$c' \leftarrow \textsc{Kyber}.\mathsf{CPAPKE.Enc}(p, m, r)$ ▷ CPA-secure re-encryption with recovered seed
if $c = c'$ **then** ▷ Equality checking
    $k \leftarrow \mathsf{KDF}(k \| \mathsf{H}(c))$ ▷ Return shared secret on success
else
    $k \leftarrow \mathsf{KDF}(\mathbf{z} \| \mathsf{H}(c))$ ▷ Implicit rejection on failure
end if

It can be derived that the decryption faces an accumulated error on the message $m \in \mathbb{R}_{(\rho,\eta)}$ as given in Equation (6), where summands $\Delta\mathbf{u}$ and Δv denote contributions from the lossy ciphertext compression. If it holds for each coefficient $i \in [0, \eta - 1]$ that $-\rho/2 < \Delta m_i < \rho/2$, then the decryption outputs the correct message $m \in \{0,1\}^\eta$ after its final step $\mathsf{Compress}(\cdot; \rho, 1)$.

$$\Delta m = \mathbf{e}^\intercal \mathbf{r} - \mathbf{s}^\intercal (\mathbf{e}_1 + \Delta\mathbf{u}) + e_2 + \Delta v \mod {}^{\pm}\rho \tag{6}$$

Barrett and Montgomery reduction methods enable efficient and time-constant implementations of modular arithmetic in a prime field $\mathbb{Z}_\rho$. In a Barrett reduction, the problematic division in $(a \mod \rho) = a - \lfloor a/\rho \rfloor \rho$ is avoided by approximating $1/\rho$ by a well-chosen constant $\gamma/2^\sigma$, given that division by a power of two is merely a shifting operation. In a Montgomery reduction, the standard remainder r such that $a = m\rho + r$ is replaced by the so-called Hensel remainder r' defined as $a = m\rho + r'\beta$, where β is the word size and $-\beta/2 \leq m < \beta/2$.

2.2 Key-encapsulation mechanism

Kyber uses a variation of the FO transform, as specified in Algorithms 4 to 6. Essentially, the ciphertext c received by the decapsulation is re-encrypted after decryption and the result c' is compared to c. If this comparison fails, the decapsulation returns a pseudorandom value instead of a failure symbol $\perp$, which is referred to as *implicit rejection*. Hash functions G and H are modeled as *random oracles* and instantiated with SHA3-512 and SHA3-256 respectively. The *key-derivation function* (KDF) is instantiated with SHAKE-256.

Algorithm 7 Montgomery [KRSS, commit on 20 Jan 2020]

Input: Integer a where $-(\beta/2) \cdot \rho \leq a < (\beta/2) \cdot \rho$ and $\beta = 2^{16}$
Input: Prime $\rho = 3329$
Input: Negated inverted prime $-\rho^{-1} = 3327$
Output: Reduced $t[31:16]$ where $-\rho < t[31:16] < \rho$

smulbb t, a, $-\rho^{-1}$ ▷ $t \leftarrow (a \mod \beta) \cdot (-\rho^{-1})$
smlabb t, ρ, t, a ▷ $t[31:16] \leftarrow \lfloor((t \mod \beta)\rho + a)/2^{16}\rfloor$

Algorithm 8 DoubleGSButterfly [KRSS, commit on 20 Jan 2020]

Input: $(a[15:0], b[15:0])$ to first butterfly
Input: $(a[31:16], b[31:16])$ to second butterfly
Input: Twiddle factor $\tau[15:0]$ or $\tau[31:16]$
Output: $(a[15:0], b[15:0])$ from first butterfly
Output: $(a[31:16], b[31:16])$ from second butterfly

usub16 t_1, a, b ▷ $\begin{cases} t_1[15:0] \leftarrow a[15:0] - b[15:0], \\ t_1[31:16] \leftarrow a[31:16] - b[31:16] \end{cases}$
uadd16 a, a, b ▷ $\begin{cases} a[15:0] \leftarrow a[15:0] + b[15:0], \\ a[31:16] \leftarrow a[31:16] + b[31:16] \end{cases}$
smulbt/smulbb b, t_1, τ ▷ $b \leftarrow t_1[15:0] \cdot \tau[\cdots]$
smultt/smultb t_1, t_1, τ ▷ $t_1 \leftarrow t_1[31:16] \cdot \tau[\cdots]$
montgomery ρ, $-\rho^{-1}$, b, t_2 ▷ Algorithm 7: reduce b to $t_2[31:16]$
montgomery ρ, $-\rho^{-1}$, t_1, b ▷ Algorithm 7: reduce t_1 to $b[31:16]$
pkhtb b, b, t_2, asr#16 ▷ $b[15:0] \leftarrow t_2[31:16]$

Algorithm 9 DoubleBarrett [KRSS, commit on 20 Jan 2020]

Input: Integers $a[15:0], a[31:16]$
Input: Prime $\rho[15:0] = 3329$
Input: Multiplicand $\gamma[15:0] = 20159$ ▷ $\begin{cases} \sigma = \lfloor \log_2(\rho) \rfloor - 1 + 16 = 26, \\ \gamma = \lfloor 2^\sigma / \rho \rceil \end{cases}$
Output: Reduced integers $a[15:0], a[31:16] \in [0, \rho - 1]$

smulbb t_1, a, γ ▷ $t1 \leftarrow a[15:0] \cdot \gamma[15:0]$
smultb t_2, a, γ ▷ $t2 \leftarrow a[31:16] \cdot \gamma[15:0]$
asr t_1, t_1, #26 ▷ $t_1 \leftarrow t_1 \gg 26$
asr t_2, t_2, #26 ▷ $t_2 \leftarrow t_2 \gg 26$
smulbb t_1, t_1, ρ ▷ $t_1 \leftarrow t_1[15:0] \cdot \rho[15:0]$
smulbb t_2, t_2, ρ ▷ $t_2 \leftarrow t_2[15:0] \cdot \rho[15:0]$
pkhbt t_1, t_1, t_2, lsl#16 ▷ $t_1[31:16] \leftarrow t_2[15:0]$
usub16 a, a, t_1 ▷ $\begin{cases} a[15:0] \leftarrow a[15:0] - t_1[15:0], \\ a[31:16] \leftarrow a[31:16] - t_1[31:16] \end{cases}$

2.3 ARM Cortex-M4

Following a recommendation by NIST, the ARM Cortex-M4 is the primary *reduced instruction set computer* (RISC) processor for benchmarking the implementation efficiency of post-quantum schemes. This embedded processor features sixteen 32-bit registers: thirteen for general purposes and three reserved for the stack pointer, the link register, and the program counter. The general-purpose registers may pack two 16-bit signed integers; instructions that perform multiplications, subtractions, and other operations on these halfwords are supported.

Source code for many schemes is publicly available in the *pqm4* library [KRSS]. Although the Kyber implementations are largely written in C, we analyze routines written in assembly exclusively. Given that prime $\rho = 3329 < 2^{12}$, 16-bit halfwords can efficiently store polynomial coefficients whilst providing a margin for lazy reductions, i.e., reductions after additions and subtractions that do not cause overflow may be skipped. As pointed out by Alkim et al. [ABCG20, Algorithm 11], Montgomery reductions can be implemented using two instructions only. Algorithm 7 shows the latest version from the *pqm4* library, which only differs from the academic paper in how temporary variables are used. The NTT and INTT exclusively rely on these Montgomery reductions, as evidenced by the double GS butterfly in Algorithm 8.

Unfortunately, the Montgomery-reduced coefficients lie in the interval $[-\rho + 1, \rho - 1]$ instead of $[0, \rho-1]$. To obtain coefficients in the interval $[0, \rho-1]$ right before compression, a slower Barrett reduction is used. As shown in Algorithm 9, a Barrett reduction on two signed coefficients requires eight instructions.

3 Side-Channel Analysis of Lattice-Based KEMs

As specified in Algorithm 6, the decryption is the only building block of Kyber's decapsulation that uses the private key $\hat{s}$ and is thus the obvious target for a SCA attack. However, SCA-assisted chosen-ciphertext attacks proposed by D'Anvers et al. [DTVV19], Ravi et al. [RRCB20], and Ueno et al. [UXT$^+$22] subverted this intuition. In many lattice-based schemes, ciphertexts c can be constructed such that the correctness of a decrypted message bit $m \in \{0, 1\}$ depends on $\hat{s}$. By exploiting a series of leakages traces of the execution of the hash function G, the encryption, or the polynomial comparison as a message-checking oracle, $\hat{s}$ is recovered. In conclusion, almost every single component of the decapsulation should be protected. The academically preferred way of countering SCA attacks is to randomize computations such that dependencies between internal secrets and measurable emissions are weakened. Below, we distinguish between masking methods, which are expensive and substantiated by a security proof in a probing model, and blinding methods, which are cheap and unsupported by a security proof.

3.1 Masking

In masked implementations, finite ring elements $x \in \mathcal{X}$ are randomly and uniformly split into $\lambda \geq 2$ shares according to Definition 1. According to Lemma 1, one way to meet Definition 1 is to first select $\left(x^{(2)}, x^{(3)}, \cdots, x^{(\lambda)}\right)$ uniformly at random from $\mathcal{X}^{\lambda-1}$, followed by a computation $x^{(1)} = x - x^{(2)} - x^{(3)} - \cdots - x^{(\lambda)}$.

Definition 1 (Uniformity)**.** A finite ring element $x \in \mathcal{X}$ is randomly and uniformly split into $\lambda \geq 2$ shares if $\Pr(x^{(1)}, x^{(2)}, \cdots, x^{(\lambda)} \mid x)$ equals $1/|\mathcal{X}|^{\lambda-1}$ if $x^{(1)} + x^{(2)} + \cdots + x^{(\lambda)} = x$ and 0 otherwise.

Lemma 1 (Subset of Shares)**.** *For a finite ring element $x \in \mathcal{X}$ that is randomly and uniformly split into λ shares according to Definition 1, any tuple of $\lambda-1$ shares is uniformly* *distributed on* $\mathcal{X}^{\lambda-1}$ *and thus independent of* x*. More generally, any tuple of* $\alpha \in [1, \lambda-1]$ *shares is uniformly distributed on* $\mathcal{X}^{\alpha}$*.*

We distinguish between Boolean masking, where $\mathcal{X} = \{0,1\}^{\sigma}$ and additions are defined by XORing, and arithmetic masking, where $x \in \mathbb{Z}_{\rho}$, and additions are performed modulo a prime ρ. For efficiency reasons, Boolean masking is typically used for symmetric-key algorithms, whereas arithmetic masking is used for polynomial operations. Hence, *Boolean-to-arithmetic* (B2A) and *arithmetic-to-Boolean* (A2B) conversions are commonplace in lattice-based cryptography.

A function $\mathsf{F} : \mathcal{X} \rightarrow \mathcal{Y}$ must also be split such that shares of $x \in \mathcal{X}$ satisfying Definition 1 are mapped to shares of $y = \mathsf{F}(x)$ that again satisfy Definition 1. If F is linear, F is trivially split by defining $\forall i \in [1, \lambda] : \mathsf{F}^{(i)}(x^{(i)}) \triangleq \mathsf{F}(x^{(i)})$, considering that $\mathsf{F}^{(1)}(x^{(1)}) + \mathsf{F}^{(2)}(x^{(2)}) + \cdots + \mathsf{F}^{(\lambda)}(x^{(\lambda)}) = \mathsf{F}(x^{(1)} + x^{(2)} + \cdots + x^{(\lambda)}) = \mathsf{F}(x)$. For lattice-based cryptography, linear components include polynomial additions, the NTT, and the INTT. Non-linear components, such as $\mathsf{Compress}$ in Equation (3) and the polynomial comparison, require custom-developed masking schemes [BGR+21].

3.2 Blinding

For blinding methods, we distinguish between randomization of data and randomization of time. The latter can be achieved by randomly permuting the order of parallelizable operations [Saa18, OSPG18, RPBC20, PP21]. For example, the polynomial coefficients fed into $\mathsf{Compress}$ in Equation (3) and $\mathsf{Decompress}$ in Equation (4) can be permuted. Similarly, the butterfly operations within an NTT/INTT layer can be shuffled.

To randomize data in a polynomial multiplication $c[x] = a[x] \cdot b[x]$ in a ring $\mathbb{R}_{(\rho,\eta)}$, Saarinen [Saa18] proposed computing $a'[x] = \alpha \cdot a[x]$ and $b'[x] = \beta \cdot b[x]$ where α and β are chosen randomly, uniformly, and independently from $\mathbb{Z}_{\rho}$, and multiplying $c'[x] = a'[x] \cdot b'[x]$ with $\gamma = (\alpha\,\beta)^{-1}$. Alternatively, the final step could be omitted by selecting α uniformly at random from $\mathbb{Z}_{\rho}$ and computing $\beta = \alpha^{-1}$, which implies $\gamma = 1$.

For an NTT-based multiplication, Saarinen [Saa18] suggested lowering the costs by computing $\alpha = \zeta^{i}$, $\beta = \zeta^{j}$, and $\gamma = \zeta^{-i-j}$, where i and j are chosen randomly, uniformly, and independently from $[0, \eta - 1]$. If a lookup table of the powers of ζ is available, numerous multiplications in $\mathbb{Z}_{\rho}$ can be avoided. Furthermore, unlike $\mathbb{Z}_{\rho}$, the cardinality of $[0, \eta - 1]$ is a power of two, which eliminates the need for *rejection sampling* given that *random number generators* output binary vectors. Ravi et al. [RPBC20] applied the latter technique at finer granularities: instead of generating blinding factors ζ^{i} for an entire polynomial multiplication, factors can be generated for individual NTT/INTT layers or even for individual butterflies. In its most generic form, the GS butterfly in Equation (2) is realized as in Equation (7), where blinding factors ζ^{i} and ζ^{j} will eventually cancel out.

$$\mathsf{BlindedGSButterfly}(a, b; \zeta^{k}) \triangleq \big((a+b)\,\zeta^{i}, (a-b)\,\zeta^{k+j}\big) \mod \rho. \tag{7}$$

4 Fault Injection against Lattice-Based KEMs

For KEMs, the decapsulation is particularly vulnerable—an attacker can fault this module a virtually unlimited number of times in order to retrieve the private key. Not surprisingly, we only target the decapsulation, even though one faulted key generation may result in a mathematically weak key pair, and even though one faulted encapsulation may result in message recovery [VOGR18, RRB+19].

4.1 Differential fault analysis

As pointed out by Oder et al. [OSPG18], a positive side effect of using the Fujisaki–Okamoto transform is that many fault attacks on the decapsulation are inherently countered: by re-encrypting the decrypted message m' and comparing the result to the externally provided ciphertext c, secret-revealing faulted data is kept internal instead of forwarded to the output. This countermeasure, which also exists in a simpler form where an encryption or decryption is executed twice, is well-established since the early 2000s, at which time Karri et al. [KWMK02] protected block ciphers such as the *Advanced Encryption Standard* (AES) against *differential fault analysis* (DFA). For block ciphers, the countermeasure can only be defeated through a double fault injection: a fault in the encryption can compensate a fault in the decryption such that the equality-check is passed, or a fault can skip the equality-check so that an arbitrary fault in the encryption propagates to the output. Unfortunately, and as surveyed by Xagawa et al. [XIU+21], the lattice-based version can be broken through a single fault that skips the equality check, considering that resistance to chosen-ciphertext attacks is not baked-in.

4.2 Ineffective Faults

Another concern is that the above countermeasure only counters DFA, or more generally, any attack that leverages faulted data. As already established in the 2000s, mere knowledge of whether or not the execution of a keyed cryptographic algorithm fails after injecting a fault can enable key recovery. Faults of the latter type are often referred to as safe errors [YJ00] or ineffective faults [Cla07]. Below, we recapitulate three applications to lattice-based cryptography.

Bettale, Montoya, and Renault [BMR21] exploited that the secret polynomials of lattice-based schemes have relatively many coefficients that are zero—if they are drawn from CBD or other small-error distributions. Hence, by setting these coefficients to zero and observing whether such faults are effective, many coefficients are revealed to be zero. Kyber, however, cannot be defeated, given that the CBD coefficients of the private key $\mathbf{s}$ are stored and used in the NTT domain in Algorithm 3, i.e., the transformed coefficients are virtually uniformly distributed on $[0, \rho - 1]$. In other schemes, shuffling and masking can preclude the attack.

Pessl and Prokop [PP21] skipped an instruction in the final compression step of Kyber's decryption, i.e., Line 5 in Algorithm 3, such that the observed effectiveness of the fault reveals the sign of the accumulated error Δm in Equation (6). By gathering thousands of these inequalities, the system can be solved for the secret $(\mathbf{s}, \mathbf{e})$ through belief propagation. This algorithm is chosen instead of linear programming for two reasons: large dimensions can be handled and occasional errors in the inequalities are tolerable.

4.3 Ineffective Faults at Indocrypt 2021

Hermelink et al. [HPP21] solved an identical system of inequalities through belief propagation, but collected the inequalities using a different method: the aforementioned SCA-assisted chosen-ciphertext attacks [DTVV19, RRCB20, UXT+22] are adapted such that the message-checking oracle is realized through fault injections instead of leakage measurements. More precisely, the attacker manipulates one coefficient v_i of the compressed ciphertext polynomial $v[x]$ of an otherwise correctly computed encapsulation by replacing Line 8 in Algorithm 2 with Equation (8), and the (in)ability to rectify the manipulation by faulting either input of the polynomial comparison reveals the sign of Δm in Equation (6). Recall that the coins r used by re-encryption are derived from the message m using a hash function, so changing a single message bit alters the entire ciphertext.

$$v^{\star}[x] = \mathsf{Compress}(v[x] + \lfloor \rho/4 \rfloor x^i). \tag{8}$$

In response to Equation (8), the accumulated error Δm faced by the decryption in Equation (6) increases by $\lfloor \rho/4 \rfloor$, as given in Equation (9).

$$\Delta m^{\star} = \Delta m + \lfloor \rho/4 \rfloor. \tag{9}$$

From Equation (9) and the observed correctness of the faulted decapsulation, an inequality follows in Equation (10). The difference in strictness is due to rounding. If m_i is correct, an attacker is able to fault coefficient v_i in either input operand of the polynomial comparison such that the decapsulation succeeds. If m_i is incorrect, any attempt for rectification is in vain.

$$m \neq m^\star \iff (m = 0 \wedge \Delta m > 0) \vee (m = 1 \wedge \Delta m \geq 0). \tag{10}$$

To condense the fault model into single bit flips, using a laser, the *Hamming distance* (HD) constraint in Equation (11) is imposed when manipulating the encapsulation. Assuming an attacker with an optimal FI setup, around 6000, 7000, and 9000 faulted decapsulations suffice to recover the private key of Kyber512, Kyber768, and Kyber1024 respectively, with a success rate of nearly 100%. If multiple bits can reliably be flipped, the HD constraint can be removed. Although the authors assumed perfect bit flips in their simulated attacks, a few trials would suffice to cover imperfect bit flips.

$$\mathsf{HD}(\mathsf{Compress}(v), \mathsf{Compress}(v[x] + \lfloor \rho/4 \rceil x^i)) = 1 \tag{11}$$

The attack may be hindered by masking, shuffling, and/or double executions, but is not precluded. Therefore, the authors proposed an additional countermeasure: instead of ciphertexts c, pairs $(c, \mathsf{Hash}(c))$ are stored in *random-access memory* (RAM) and eventually compared. Although faulting c while it is stored in RAM becomes pointless, the attack still succeeds by faulting c before it is fed into the hash function, e.g., in the back end of $\mathsf{Compress}(v; \rho, \delta_v)$.

5 Roulette Attacks

We first present the general attack methodology in Section 5.1, and apply this methodology to Kyber's decapsulation in Section 5.2.

5.1 General Methodology

Consider a keyed cryptographic algorithm $\mathsf{A} : \mathcal{S} \times \mathcal{I} \to \mathcal{O}$ where $s \in \mathcal{S}$ is keying material, $i \in \mathcal{I}$ is the public input, and $o \in \mathcal{O}$ is the output. Output o is not necessarily public, but an attacker can observe whether or not o is correct. We decompose A into five parts, as shown in Figure 1.

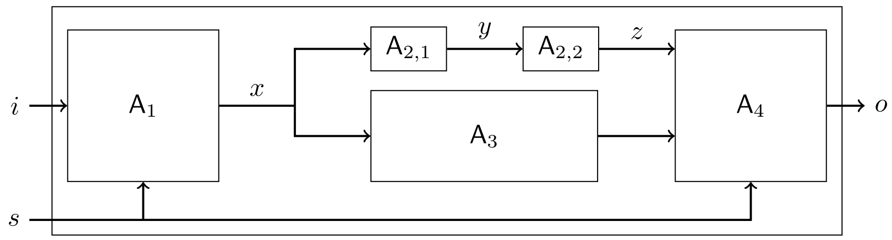

Figure 1: Decomposition of cryptographic algorithm A.

To keep the execution time of the attack within bounds, we require that cardinalities $|\mathcal{Y}|$ and $|\mathcal{Z}|$ are small. For a constant input (s, i), the attacker repeatedly faults either $\mathsf{A}_{2,1}$ or y or $\mathsf{A}_{2,2}$ or z such that $z^\star \in \mathcal{Z}$ is not constant, i.e., $z^\star$ does not follow a one-point distribution with respect to the infinite set of fault injections. Although many distributions might enable an attack, we idealize the case where $z^\star$ is uniformly distributed on $\mathcal{Z}$. In our casino analogy, this corresponds to spinning a roulette wheel, at least if we visualize

$\mathcal{Z}$ through a circular representation. This analogy also emphasizes that random draws are an essential element of the attack. If for the given distribution of $z^\star$, the probability that A fails to produce the correct output o depends on the secret $s \in \mathcal{S}$, then the attacker can retrieve information on s.

Our motivation for idealizing (nearly) uniform distributions of $z^\star \in \mathcal{Z}$ is that they naturally support (i) a large attack surface and (ii) various fault models, especially when SCA countermeasures such as masking and data-randomizing blinding are deployed. Section 5.1.1 formalizes the notion that uniformly distributed faults tend to propagate to uniformly distributed faults. Section 5.1.2 gives examples of supported fault models.

5.1.1 Attack Surface

For a function that is balanced according to Definition 2, uniformly distributed faults propagate as uniformly distributed faults, as formalized in Lemma 2 and proven in Appendix C.1. If the function $A_{2,2} : \mathcal{Y} \to \mathcal{Z}$ in Figure 1 happens to be balanced, an attacker who is able fault $A_{2,1}$ or y such that the faulted value $y^\star \sim U(\mathcal{Y})$, indirectly achieves $z^\star \sim U(\mathcal{Z})$.

Definition 2 (Balanced Function)**.** Let $\mathsf{F} : \mathcal{A} \to \mathcal{C}$ be a function. If it holds $\forall c \in \mathcal{C}$ that $|\{a \in \mathcal{A} \mid \mathsf{F}(a) = c\}| = |\mathcal{A}|/|\mathcal{C}|$, then F is balanced. Similarly, for a function $\mathsf{F} : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$, if it holds $\forall (b, c) \in \mathcal{B} \times \mathcal{C}$ that $|\{a \in \mathcal{A} \mid \mathsf{F}(a, b) = c\}| = |\mathcal{A}|/|\mathcal{C}|$, then F is balanced with respect to input $a \in \mathcal{A}$.

Lemma 2 (Fault Propagation for Balanced Functions)**.** *Let $\mathsf{F} : \mathcal{A} \to \mathcal{C}$ be a balanced function, as formalized in Definition 2. If $a \sim U(\mathcal{A})$, then $c \sim U(\mathcal{C})$. Similarly, for a function $\mathsf{F} : \mathcal{A} \times \mathcal{B} \to \mathcal{C}$ that is balanced with respect to input $a \in \mathcal{A}$, if $a \sim U(\mathcal{A})$ is independent of $b \in \mathcal{B}$, then $c \sim U(\mathcal{C})$.*

Fortunately for the attacker, balanced functions are frequently used in cryptography. Bijections are a trivial example. Addition in a finite ring and multiplication in a finite field are two more examples, as formalized in Lemmas 3 and 4 respectively, and proven in Appendices C.2 and C.3 respectively. In fact, balancedness is merely the ideal case; imbalanced fault propagation may still enable an attack in practice.

Lemma 3 (Balancedness of Addition in Finite Ring)**.** *Let $\mathcal{R}$ be a finite ring and let $\mathsf{F} : \mathcal{R}^2 \to \mathcal{R}$ be defined as $c \triangleq \mathsf{F}(a, b) \triangleq a + b$. It holds that F is fully balanced, i.e., Definition 2 is met with respect to both input $a \in \mathcal{R}$ and $b \in \mathcal{R}$.*

Lemma 4 (Balancedness of Multiplication in Finite Field)**.** *Let $\mathcal{F}$ be a finite field and let $\mathsf{F} : \mathcal{F}^2 \to \mathcal{F}$ be defined as $c \triangleq \mathsf{F}(a, b) \triangleq a \cdot b$, where $b \neq 0$. It holds that F is balanced, i.e., Definition 2 is met with respect to $a \in \mathcal{F}$.*

5.1.2 Fault Models

Examples 1 to 4 demonstrate that the ideal distribution, $z^\star \sim U(\mathcal{Z})$, can be achieved for various fault models. Despite assuming that the attacker faults either $A_{2,2}$ or z, balanced fault-propagation properties according to Section 5.1.1 may extend the attack surface to $A_{2,1}$ and y. Again, note that a uniform distribution is merely the ideal case; other distributions may enable an attack as well.

Example 1 (Random Faults)**.** Random faults where $z^\star \sim U(\mathcal{Z})$ comprise a well-established fault model in the academic literature and are covered by definition. Also stronger fault models where $z \in \{0,1\}^\lambda$ is XORed with an attacker-chosen error $e \in \{0,1\}^\lambda$ are covered. If the attacker chooses $e \sim U(\{0,1\}^\lambda)$, then $z^\star \triangleq z \oplus e \sim U(\{0,1\}^\lambda)$.

Example 2 (Set-To-Constant Faults)**.** Set-to-0 and set-to-1 faults are covered for masked implementations. Let z be randomly and uniformly split into $\lambda \geq 2$ shares according to Definition 1, and without loss of generality, assume that the first share, $z^{(1)} \in \mathcal{Z}$, is set to an arbitrary constant $\theta \in \mathcal{Z}$, whereas shares $z^{(2)}, \cdots, z^{(\lambda)} \in \mathcal{Z}$ are untouched. Considering that $z^{(1)} \sim U(\mathcal{Z})$ and $\left(z^{(2)}, \cdots, z^{(\lambda)}\right) \sim U(\mathcal{Z}^{\lambda-1})$ according to Lemma 1, it follows that the faulted value $z^\star = \theta + z^{(2)} + \cdots + z^{(\lambda)} = z - z^{(1)} + \theta \sim U(\mathcal{Z})$.

Example 3 (Instruction Skips)**.** Let $\mathsf{A}_{2,2} : \mathcal{Y} \rightarrow \mathcal{Z}$ be realized through a masked software implementation. Without loss of generality, assume that an instruction in the first share function, $\mathsf{A}_{2,2}^{(1)}$, is skipped such that the faulty output share $(z^{(1)})^\star$ is independent of the correct output share $z^{(1)}$. Hence, $z^\star = (z^{(1)})^\star + z^{(2)} + \cdots + z^{(\lambda)}$ is again uniformly distributed on $\mathcal{Z}$.

Example 4 (Arbitrary Bit Flips)**.** Let $\mathsf{A}_{2,2} : \mathcal{Y} \rightarrow \mathcal{Z}$ be an affine function over a finite field $\mathcal{Y} = \mathcal{Z} = \{0,1\}^\lambda$ where addition is defined by XORing. Let $z \triangleq \mathsf{A}_{2,2}(y)$ be realized through a blinded implementation $z = r^{-1}\,\mathsf{A}_{2,2}(r \cdot y)$ where $r \sim U(\{0,1\}^\lambda \setminus \{0\})$. For any pattern of bit flips $e \in \{0,1\}^\lambda \setminus \{0\}$ applied to the input of $\mathsf{A}_{2,2}$, it holds that the faulted output $z^\star \triangleq r^{-1}\,\mathsf{A}_{2,2}(r \cdot y \oplus e) = z \oplus r^{-1}\,\mathsf{A}_{2,2}(e) \sim U(\{0,1\}^\lambda \setminus \{0\})$. Strictly speaking, this distribution is nearly uniform, given that the case $z^\star = z$ is excluded. One could achieve $z^\star \sim U(\{0,1\}^\lambda)$ by aborting the fault injection with probability $1/2^\lambda$, but this would be pointless in an actual attack.

5.1.3 Comparisons

Table 1 compares our Roulette attacks to well-known fault attacks, i.e., DFA, FSA [LOS12], and SIFA [DEK+18]. The standout property of Roulette attacks is that masking is a facilitator. Although masking may not preclude DFA [BH08], FSA [MMP+11, Del20], or SIFA [DEG+18], it is not a facilitator here. Furthermore, note that the fault distributions of Roulette and SIFA are complementary to some extent.

Table 1: Comparison of fault attacks.

Technique	**DFA**	**FSA**	**SIFA**	**Roulette**
Input i	Unknown	Known	Unknown	Known
Correct output o	Known	Unknown	Known	Unknown
Faulty output $o^\star$	Known	Unknown	Unknown	Unknown
Input i	Constant i	Constant i	$i \leftarrow U(\mathcal{I})$	Constant i
Correct intermediate z	Constant z	Constant z	$z \sim U(\mathcal{Z})$	Constant z
Faulty intermediate $z^\star$	Any	Any	$z^\star \not\sim U(\mathcal{Z})$	Any, $z^\star \sim U(\mathcal{Z})$
Fault intensity	Constant	Variable	Constant	Constant
Masking	Nuisance	Nuisance	Nuisance	Facilitator
Duplication	Game over	Don't care	Don't care	Don't care

5.2 Application to Kyber's Decapsulation

We now instantiate the generic cryptographic algorithm A from Section 5.1 with Kyber's decapsulation, as specified in Algorithm 6. Our first and foremost Roulette attack is an extension of the attack of Hermelink et al. [HPP21]; the private key s is recovered by faulting the re-encryption. A second Roulette attack recovers the message m and the corresponding session key k by faulting the decryption. Considering that the second attack is far less practical while recovering the short-term and thus not the long-term secret, its specification is deferred to Appendix B.

5.2.1 Attack Surface

The generic variable $z \in \mathcal{Z}$ in Figure 1 is instantiated with a compressed ciphertext coefficient $v \in \{0,1\}^{\delta_v}$ that is output from the re-encryption, as specified in Algorithm 2. Following Hermelink et al. [HPP21], the goal is to match a manipulated coefficient so that the polynomial comparison succeeds, at least if the preceding decryption is correct. If the faulted value $v^\star$ is uniformly distributed on $\{0,1\}^{\delta_v}$, then the probability of a successful decapsulation, P_{success}, is approximately 0 if $m \neq m^\star$ and $1/2^{\delta_v}$ otherwise. The probability that at least one out of $n \in \mathbb{N}_0$ faulted decapsulations is successful becomes $1-(1-1/2^{\delta_v})^n$ in the latter case and we arbitrarily impose $P_{\text{success}} \geq 99\%$, considering that belief propagation is somewhat error-tolerant. For Kyber512 and Kyber768, where $\delta_v = 4$, this implies $n \geq 72$. For Kyber1024, where $\delta_v = 5$, this implies $n \geq 146$. The attacker always performs 72 or 146 injections if $m \neq m^\star$, but can stop prematurely after the first success otherwise.

Compared to the attack of Hermelink et al. [HPP21] in its original form, the number of fault injections increases by roughly one or two orders of magnitude, but we get a considerably larger attack surface and support for various fault models in return. As illustrated in Figure 2, the function $A_2 \triangleq A_{2,2} \circ A_{2,1}$ that produces a coefficient $v \in \{0,1\}^{\delta_v}$ comprises one GS butterfly in the last layer of an INTT, the generation of one CBD sample, the decompression of one message bit, one modular addition, and one compression. Moreover, by faulting any of these building blocks, the countermeasure of Hermelink et al. [HPP21] to store $(c, \mathsf{Hash(c)})$ in RAM is bypassed.

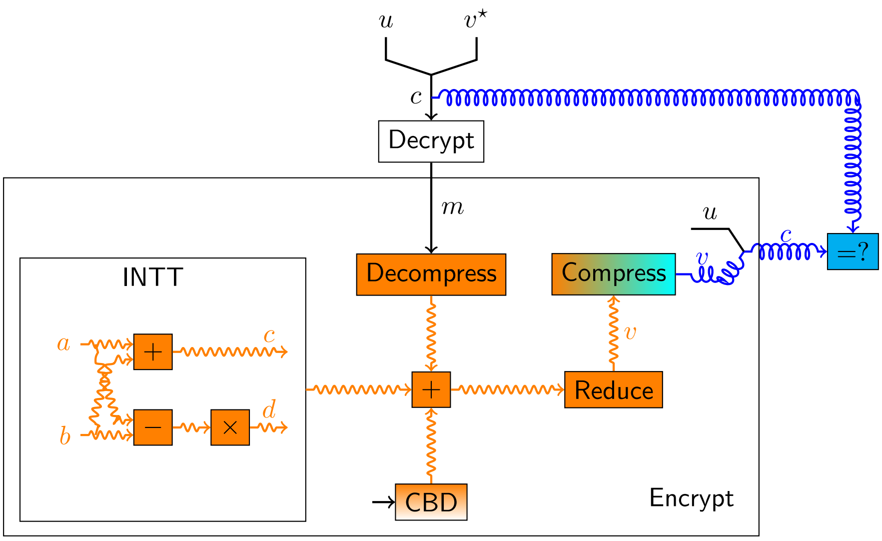

Figure 2: The attack surface of Hermelink et al. [HPP21] is colored blue; our extension is colored orange.

Another godsend for the attacker is that the fault-propagation statistics are almost ideal. The modular addition is perfectly balanced according to Definition 2 with respect to all three inputs (this is a trivial generalization of Lemma 3). Ciphertext compression as defined in Equation (3) is not perfectly balanced, but the deviation is too small to notably impact the attack. If we introduce faults such that the uncompressed coefficient is uniformly distributed on $[0, \rho-1]$, then the compressed coefficient slightly deviates from uniform. For $\delta = 4$, the zero coefficients occurs with probability $209/3329$, whereas all other coefficients occur with probability $208/3329$. Similarly, for $\delta = 5$, this becomes $105/3329$ for the zero coefficient and $104/3329$ for all other coefficients.

5.2.2 Optional Hamming-Distance Constraint

The sole purpose of the Hamming distance constraint in Equation (11) is to establish single bit flips as the fault model. In our extension of the attack, this constraint does not affect the feasibility of a fault injection and is thus entirely optional. To accommodate a potential omission, we extend Equations 8 to 10. As a starting point, we summarize the behavior of Compress in Equation (3) and Decompress in Equation (4). For KYBER512 and KYBER768, where $\delta_v = 4$, our summary is contained in the first five columns of Table 2. For brevity, we do not discuss KYBER1024, where $\delta_v = 5$, but identical conclusions can be drawn from Table 4 in Appendix A.

Table 2: Properties of the compressed ciphertext coefficients $v \in [0, 2^{\delta} - 1]$ where $\delta = 4$. The first and last elements of each bin are defined by Compress in Equation 3. The bin centers are defined by Decompress in Equation 4.

Original					Manipulated			
Bin	Size	First	Last	Center	Bin	Fault	HD	$\Delta m^{\star}$
0	209	3225	104	0	4			
1	208	105	312	208	5	0100	1	$\Delta m + 832$
2	208	313	520	416	6	0100	1	$\Delta m + 832$
3	208	521	728	624	7	0100	1	$\Delta m + 832$
4	208	729	936	832	8	1100	2	$\Delta m + 833$
5	208	937	1144	1040	9	1100	2	$\Delta m + 833$
6	208	1145	1352	1248	10	1100	2	$\Delta m + 833$
7	208	1353	1560	1456	11	1100	2	$\Delta m + 833$
8	208	1561	1768	1665	12	0100	1	$\Delta m + 832$
9	208	1769	1976	1873	13	0100	1	$\Delta m + 832$
10	208	1977	2184	2081	14	0100	1	$\Delta m + 832$
11	208	2185	2392	2289	15	0100	1	$\Delta m + 832$
12	208	2393	2600	2497	0	1100	2	$\Delta m + 833$
13	208	2601	2808	2705	1	1100	2	$\Delta m + 833$
14	208	2809	3016	2913	2	1100	2	$\Delta m + 833$
15	208	3017	3224	3121	3	1100	2	$\Delta m + 833$

An evident anomaly is that bin 0 is 'oversized': it contains 209 elements, whereas 15 'ordinary' bins each contain 208 elements. The proposed manipulation in Equation (8) is to add $\lfloor \rho/4 \rfloor = 832 = 4 \cdot 208$ to the uncompressed coefficient, which is a jump spanning exactly 4 'ordinary' bins. Unfortunately, the first element of bin 0 then maps to the last element of bin 3, given that $3225 + 832 \mod 3329 = 728$, and thus not to the first element of bin 4. In absence of the HD constraint in Equation (11), the decryption would face an accumulated error $\Delta m^{\star} = \Delta m + 632$, which significantly undershoots the desired effect $\Delta m^{\star} = \Delta m + 832$ in Equation (9). An easy fix is to replace Equation (8) by a direct manipulation of the compressed coefficient, as given in Equation (12).

$$v^{\star} = v + 2^{\delta_v - 2} \mod 2^{\delta_v}. \tag{12}$$

Furthermore, in cases where the HD is 2 instead of 1, the accumulated error Δm happens to be increased by 833 instead of 832. The required extension of Equations (9) and (10) is given in Equation (13).

$$\Delta m^\star = \Delta m + 832 \implies \begin{array}{c} m \neq m^\star \iff \\ (m = 0 \wedge \Delta m \geq 1) \vee (m = 1 \wedge \Delta m \geq 0), \end{array} \tag{13a}$$

$$\Delta m^\star = \Delta m + 833 \implies \begin{array}{c} m \neq m^\star \iff \\ (m = 0 \wedge \Delta m \geq 0) \vee (m = 1 \wedge \Delta m \geq -1). \end{array} \tag{13b}$$

5.2.3 Masked Software on ARM Cortex-M4

Due to the large attack surface in Fig. 2, where most building blocks come with a plethora of implementation strategies and masking schemes, we cannot possibly be exhaustive in our demo attacks. Firstly, we consider a segment of masked software on the ARM Cortex-M4. Although the Kyber implementations in the *pqm4* library [KRSS] are unprotected, we focus on linear functions exclusively so that masking is realized merely by executing the corresponding code segments $\lambda \geq 2$ times on their respective shares. More specifically, we focus on linear functions that are written in assembly so that differences among C compilers and build settings are irrelevant. Firstly, we analyze the double GS butterfly in the last layer of the INTT, as implemented in Algorithm 8 and executed on $\lambda \geq 2$ shares. For all nine instructions, Table 3 summarizes the effect of skipping that particular instruction for a single share.

Table 3: The impact of an instruction skip on the double GS butterfly in Algorithm 8 where one out of $\lambda \geq 2$ shares is targeted. A checkmark (✓) denotes the correct result.

	Skipped instruction	$c_1^\star$	$d_1^\star$	$c_2^\star$	$d_2^\star$	Proof
1	usub16 t_1, a, b	✓	$\sim U(\mathbb{Z}_\rho)$	✓	$\sim U(\mathbb{Z}_\rho)$	Eq. (17)
2	uadd16 a, a, b	$\sim U(\mathbb{Z}_\rho)$	✓	$\sim U(\mathbb{Z}_\rho)$	✓	Eq. (14)
3	smulbb b, t_1, τ	✓	$\sim U(\mathbb{Z}_\rho)$	✓	✓	Eq. (15)
4	smultb t_1, t_1, τ	✓	✓	✓	$\sim U(\mathbb{Z}_\rho)$	Eq. (16)
5.1	smulbb t_2, b, $-\rho^{-1}$	✓	$\not\sim U(\mathbb{Z}_\rho)$	✓	✓	Fig. 3(a)
5.2	smlabb t_2, ρ, t_2, b	✓	$\not\sim U(\mathbb{Z}_\rho)$	✓	✓	Fig. 3(c)
6.1	smulbb b, t_1, $-\rho^{-1}$	✓	✓	✓	$\not\sim U(\mathbb{Z}_\rho)$	Fig. 3(b)
6.2	smlabb b, ρ, b, t_1	✓	✓	✓	$\not\sim U(\mathbb{Z}_\rho)$	Fig. 3(c)
7	pkhtb b, b, t_2, asr#16	✓	$\sim U(\mathbb{Z}_\rho)$	✓	✓	Eq. (18)

Clearly, the attacker is in a privileged position: for five out of nine instruction skips, the faulted output coefficients are uniformly distributed, which is our ideal-case scenario. For the first two instruction skips though, two output coefficients are disturbed, which implies that the attacker must perform more fault injections. To prove uniformity, we start from the observation that for each out of λ shares, the input to last INTT layer is uniformly distributed on $\mathbb{Z}_\rho^\eta = \mathbb{Z}_\rho^{256}$, which implies that all 256 finite-field elements are independent of one another. This follows from Lemma 1 and the fact that every INTT layer is a permutation on $\mathbb{Z}_\rho^\eta$. The uniformity proofs are all instances of Example 3. The proofs for instructions 2 to 4 in Eqs. (14) to (16) respectively are particularly straightforward. Note that the faulty output shares are low in magnitude even before being reduced by the Montgomery macro and cannot violate the margin for lazy reductions in any building block following the double butterfly. Also, note that the multiplications with τ, $(\tau + 1)$, or $(1 - \tau)$ preserve uniformity according to Lemma 4.

$$\begin{cases} \big(c_1^{(1)}, c_2^{(1)}\big)^\star = \big(a_1^{(1)}, a_2^{(1)}\big), \\ \big(c_1^{(2)}, c_2^{(2)}\big) = \big(a_1^{(2)} + b_1^{(2)}, a_2^{(2)} + b_2^{(2)}\big), \\ \quad \vdots \\ \big(c_1^{(\lambda)}, c_2^{(\lambda)}\big) = \big(a_1^{(\lambda)} + b_1^{(\lambda)}, a_2^{(\lambda)} + b_2^{(\lambda)}\big), \end{cases} \implies \big(c_1, c_2\big)^\star = (c_1, c_2) - \big(b_1^{(1)}, b_2^{(1)}\big) \sim U(\mathbb{Z}_\rho^2). \tag{14}$$

$$\begin{cases} \big(d_1^{(1)}\big)^\star = b_1^{(1)}, \\ d_1^{(2)} = \big(a_1^{(2)} - b_1^{(2)}\big)\tau, \\ \quad \vdots \\ d_1^{(\lambda)} = \big(a_1^{(\lambda)} - b_1^{(\lambda)}\big)\tau, \end{cases} \implies d_1^\star = d_1 + b_1^{(1)}(\tau+1) - a_1^{(1)}\tau \sim U(\mathbb{Z}_\rho). \tag{15}$$

$$\begin{cases} \big(d_2^{(1)}\big)^\star = a_2^{(1)} - b_2^{(1)}, \\ d_2^{(2)} = \big(a_2^{(2)} - b_2^{(2)}\big)\tau, \\ \quad \vdots \\ d_2^{(\lambda)} = \big(a_2^{(\lambda)} - b_2^{(\lambda)}\big)\tau \end{cases} \implies d_2^\star = d_2 + \big(a_2^{(1)} - b_2^{(1)}\big)(1-\tau) \sim U(\mathbb{Z}_\rho). \tag{16}$$

For instruction 1 in Table 3, the faulted output coefficients $(d_1, d_2)^\star$ are determined by an uninitialized temporary variable t_1, as formalized in Eq. (17). Following the INTT implementation of the *pqm4* library, each layer is completed before starting the next one, and for the most part, t_1 has last been set in another double butterfly in the last layer. Hence, t_1 is independent of the current double-butterfly inputs. As for instructions 2 to 4, the faulted output shares are reduced by the Montgomery macro.

$$\begin{cases} \big(d_1^{(1)}, d_2^{(1)}\big)^\star = \big(t_1[15:0], t_1[31:16]\big)\,\tau, \\ \big(d_1^{(2)}, d_2^{(2)}\big) = \big(a_1^{(2)} - b_1^{(2)}, a_2^{(2)} - b_2^{(2)}\big)\,\tau, \\ \quad \vdots \\ \big(d_1^{(\lambda)}, d_2^{(\lambda)}\big) = \big(a_1^{(\lambda)} - b_1^{(\lambda)}, a_2^{(\lambda)} - b_2^{(\lambda)}\big)\,\tau, \end{cases} \quad \begin{array}{r} \text{where } t_1 \text{ and} \\ \big(a_1^{(1)}, b_1^{(1)}, a_2^{(1)}, b_2^{(1)}\big) \\ \text{are independent,} \end{array} \tag{17a}$$

$$\implies \begin{array}{r} \big(d_1, d_2\big)^\star = (d_1, d_2) + \big(t_1[15:0] - a_1^{(1)} + b_1^{(1)}, \\ t_1[31:16] - a_2^{(1)} + b_2^{(1)}\big)\,\tau \sim U(\mathbb{Z}_\rho^2) \end{array} \tag{17b}$$

For instruction 7 in Table 3, the faulty output coefficient $d_1^\star$ is uniformly distributed on $\mathbb{Z}_\rho$ in theory, but not necessarily in practice. The output of the function M is not properly reduced, and the margin for lazy reduction may be violated in building blocks following the double butterfly. Such violations may still produce the desired result in Eq. (12), but are hard to analyze from a mathematical perspective and not further addressed here.

$$\begin{cases} \left(d_1^{(1)}\right)^\star = \mathsf{M}\Big(\big(a_2^{(1)} - b_2^{(1)}\big)\tau\Big), \\ d_1^{(2)} = \big(a_1^{(2)} - b_1^{(2)}\big)\tau, \\ \quad\vdots \\ d_1^{(\lambda)} = \big(a_1^{(\lambda)} - b_1^{(\lambda)}\big)\tau \end{cases} \implies \begin{aligned} d_1^\star = d_1 + \mathsf{M}\Big(\big(a_2^{(1)} - b_2^{(1)}\big)\tau\Big) \\ -\big(a_1^{(1)} - b_1^{(1)}\big)\tau \sim U(\mathbb{Z}_\rho). \end{aligned} \tag{18}$$

For instructions 5.1 to 6.2 in Table 3, a tractable closed-form expression for the distribution of the faulted coefficient $d^\star$ might not exist. Therefore, we take an empirical approach by measuring the distribution of $d^\star$ on the ARM Cortex-M4 of an STM32F407 Discovery board. An instruction skip is trivially realized by removing that particular instruction from the source code. For convenience, we measure the absolute error Δd as defined in Equation (19); this quantity is fully determined by one share and is thus independent of the correct unshared value d.

$$\Delta d \triangleq d^\star - d \mod \rho = \left(d^{(1)}\right)^\star - d^{(1)} \mod \rho. \tag{19}$$

In Figure 3, we show histograms of Δd with 64 equidistant bins for 10^4 double-butterfly inputs $(a_1^{(1)}, b_1^{(1)}, a_2^{(1)}, b_2^{(1)}) \sim U([-\rho+1, \rho-1]^4)$ and twiddle factor $\tau = 3042$. For skips of instruction 5.1 in particular, a dummy double-butterfly where $\tau = 3127$ is performed in advance on random inputs so that the temporary variable t_2 is properly initialized. The histograms for instructions 5.1 and 6.1 in Figure 3(a) and Figure 3(b) respectively are fairly uniform and thus exploitable. The histogram for instructions 5.2 and 6.2 in Figure 3(b) has empty bins around $\Delta d = \lfloor \rho/4 \rceil$ and cannot be exploited.

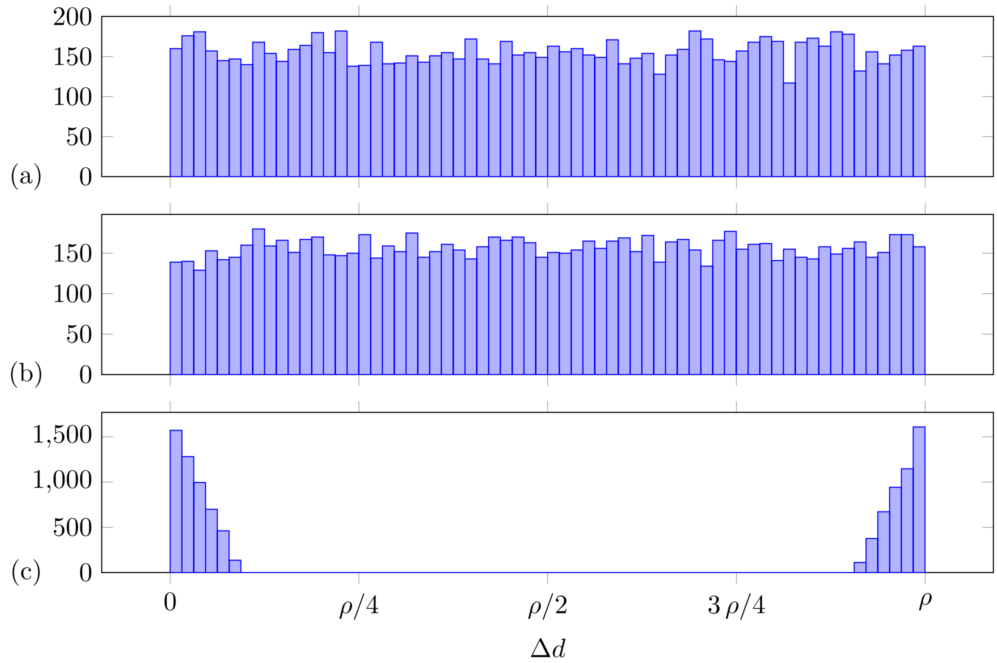

Figure 3: Histogram of double-butterfly error Δd in Equation (19) measured on an STM32F407 Discovery board for instructions (a) 5.1, (b) 6.1, and (c) 5.2 and 6.2 in Table 3.

Secondly, consider the Barrett reduction in Algorithm 9 prior to ciphertext-coefficient compression. As empirically validated on the STM32F407 Discovery board, for six out of eight instructions, the effect of a skip is that the output is not properly reduced, i.e., the absolute error is an integer multiple of prime ρ. Although the desired offset $\rho/4$ cannot be created, the attack might still succeed given that the masked compression function is likely designed to only accept inputs in the range $[0, \rho]$ and might generate all kinds of unexpected outputs outside of the this interval. The two shift instructions (`asr`) are inherently exploitable due to a higher degree of diffusion. Even with inputs in the interval $[-2\rho+1, 2\rho-1]$, the faulted output is fairly uniform across the entire 16-bit integer range. A histogram across $[-10\rho, 10\rho]$ with 320 equidistant bins is shown in Figure 4.

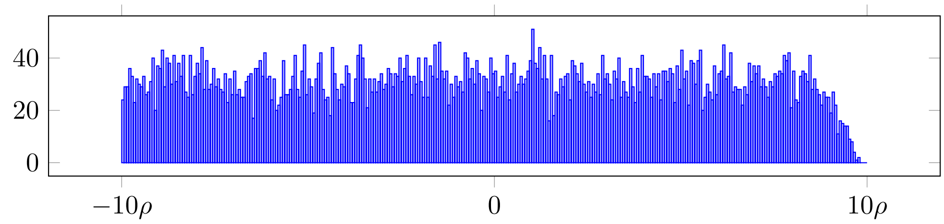

Figure 4: Histogram of Barrett error Δv measured on an STM32F407 Discovery board for instructions 3 and 4 in Algorithm 9.

5.2.4 Blinded Hardware

For attacks on hardware implementations, spatially localized fault-injections methods such as laser or electromagnetic are of particular interest. A potential target is, for example, a GS butterfly blinded according to Equation (7) in the final INTT layer. As formalized in Equation (20), if the attacker flips an arbitrary set of bits in multiplicand $(a + b)$, then the faulted butterfly output $c^\star$ is uniformly distributed on a subset of $\mathbb{Z}_\rho$ with cardinality η, given that ζ is the η-th root of unity. Contrary to Example 4, only $\eta/\rho \approx 7.7\%$ of all possible values are covered, but the attack succeeds considering that one or more values around $\Delta c = \lfloor \rho/4 \rfloor$ suffice.

$$(a+b)^\star \triangleq (a+b) \oplus e \implies \Delta c \triangleq c^\star - c = \left(\sum_{n=0}^{\lfloor \log_2(\rho) \rfloor} e[n](-1)^{(a+b)[n]} 2^n \right) \zeta^i. \tag{20}$$

Similarly, bit flips in multiplicand $(a - b)$ cause butterfly output d to be uniformly distributed on a subset of η elements in $\mathbb{Z}_\rho$. It also possible to flip bits of either a or b, but then more injections must be performed considering that c and d are simultaneously faulted.

6 Concluding Remarks

This work reveals a novel trade-off for the fault attack of Hermelink et al. [HPP21]. In exchange for more faults, the attack surface can be increased and more fault models can be covered. This also shows that a hash-based countermeasure purely for the polynomial comparison is not enough. Lastly, we suggest two directions for further work.

Firstly, other post-quantum schemes could be investigated. Hermelink et al. [HPP21] demonstrated their fault attack on Kyber [ABD+20] but conjectured that a similar attack applies to Saber [BMD+20], which is another lattice-based KEM and round-3 finalist. Similarly, we conjecture that our Roulette attacks can be mapped to Saber too. In the ideal case, a ciphertext coefficient $c_m \in \{0, 1\}^\tau$, where τ equals 3, 4, and 6 for LightSaber, Saber, and FireSaber respectively, is faulted such that $c_m^\star$ is uniformly distributed on

$\{0,1\}^\tau$. Furthermore, $c_m^\star$ is the result of rounding (pruning least-significant bits) and an addition, both of which are balanced functions as defined in Definition 2, i.e., the attack surface is large once again.

Secondly, the vulnerability analysis is performed manually in this article. Considering that novel masking and blinding implementations are continuously being proposed, it seems worthwhile to investigate to which extent our analysis can be automated.

References

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R,M}LWE schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(3):336–357, 2020.

[ABD+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: Algorithm specifications and supporting documentation. Technical report, National Institute of Standards and Technology (NIST), October 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(4):173–214, 2021.

[BH08] Arnaud Boscher and Helena Handschuh. Masking does not protect against differential fault attacks. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *5th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2008)*, pages 35–40. IEEE, August 2008.

[BMD+20] Andrea Basso, Jose Maria Bermudo Mera, Jan-Pieter DAnvers, Angshuman Karmakar, Sujoy Sinha Roy, Michiel Van Beirendonck, and Frederik Vercauteren. SABER: Mod-LWR based KEM (round 3 submission). Technical report, National Institute of Standards and Technology (NIST), October 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[BMR21] Luk Bettale, Simon Montoya, and Guénaël Renault. Safe-error analysis of post-quantum cryptography mechanisms. Cryptology ePrint Archive, Report 2021/1339, October 2021. https://ia.cr/2021/1339.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342. Springer, December 2018.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, 2018.

[Del20] Jeroen Delvaux. Threshold implementations are not provably secure against fault sensitivity analysis. Cryptology ePrint Archive, Report 2020/400, 2020. https://ia.cr/2020/400.

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum schemes. In Begül Bilgin, Svetla Petkova-Nikova, and Vincent Rijmen, editors, *Proceedings of ACM Workshop on Theory of Implementation Security Workshop (TIS@CCS 2019)*, pages 2–9. ACM, November 2019.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on Kyber. In *Progress in Cryptology - INDOCRYPT 2021*, Lecture Notes in Computer Science, page 25. Springer, 2021. https://ia.cr/2021/1222.

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

[KWMK02] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1509–1517, December 2002.

[LOS12] Yang Li, Kazuo Ohta, and Kazuo Sakiyama. New fault-based side-channel attack using fault sensitivity. *IEEE Transactions on Information Forensics and Security*, 7(1):88–97, February 2012.

[MMP+11] Amir Moradi, Oliver Mischke, Christof Paar, Yang Li, Kazuo Ohta, and Kazuo Sakiyama. On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting. In Bart Preneel and Tsuyoshi Takagi, editors, *13th Workshop on Cryptographic Hardware and Embedded Systems (CHES 2011)*, volume 6917 of *Lecture Notes in Computer Science*, pages 292–311. Springer, October 2011.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked ring-LWE implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):142–174, 2018.

[PP21] Peter Pessl and Lukás Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(2):37–60, 2021.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering - 10th International Conference, SPACE 2020, Kolkata, India, December 17-21, 2020, Proceedings*, volume 12586 of *Lecture Notes in Computer Science*, pages 123–146. Springer, 2020.

[RRB+19] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once - practical fault attack on pqm4 implementations of NIST candidates. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 232–250. Springer, 2019.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(3):307–335, 2020.

[Saa18] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 8(1):71–84, 2018.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2022.

[VOGR18] Felipe Valencia, Tobias Oder, Tim Güneysu, and Francesco Regazzoni. Exploring the vulnerability of R-LWE encryption to fault attacks. In John Goodacre, Mikel Luján, Giovanni Agosta, Alessandro Barenghi, Israel Koren, and Gerardo Pelosi, editors, *Fifth Workshop on Cryptography and Security in Computing Systems (CS2 2018)*, pages 7–12. ACM, January 2018.

[XIU+21] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In *Advances in Cryptology – ASIACRYPT 2021*, Lecture Notes in Computer Science. Springer, 2021.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

A Omitting HD Constraint in Kyber1024

Table 4: Properties of the compressed ciphertext coefficients $v \in [0, 2^{\delta} - 1]$ where $\delta = 5$.

Original					Manipulated			
Bin	Size	First	Last	Center	Bin	Fault	HD	$\Delta m^{\star}$
0	105	3277	52	0	8			
1		53	156	104	9			
2		157	260	208	10			
3		261	364	312	11	01000	1	$\Delta m + 832$
4		365	468	416	12			
5		469	572	520	13			
6		573	676	624	14			
7		677	780	728	15			
8		781	884	832	16			
9		885	988	936	17			
10		989	1092	1040	18			
11		1093	1196	1144	19	11000	2	$\Delta m + 833$
12		1197	1300	1248	20			
13		1301	1404	1352	21			
14		1405	1508	1456	22			
15		1509	1612	1560	23			
16	104	1613	1716	1665	24			
17		1717	1820	1769	25			
18		1821	1924	1873	26			
19		1925	2028	1977	27	01000	1	$\Delta m + 832$
20		2029	2132	2081	28			
21		2133	2236	2185	29			
22		2237	2340	2289	30			
23		2341	2444	2393	31			
24		2445	2548	2497	0			
25		2549	2652	2601	1			
26		2653	2756	2705	2			
27		2757	2860	2809	3	11000	2	$\Delta m + 833$
28		2861	2964	2913	4			
29		2965	3068	3017	5			
30		3069	3172	3121	6			
31		3173	3276	3225	7			

B Roulette Attack on Decryption Module

Section 5.2 specified a first roulette attack on KYBER's decapsulation, in which the re-encryption is faulted in order to recover the private key s. This appendix specifies a second roulette attack on the decapsulation, but now the decryption is faulted in order to recover the message m and the corresponding session key k. This second attack is much more 'academic' because (i) the distribution of the faulted value must be known, and (ii) millions of perfectly injected faults are required. Nevertheless, there is no harm in reporting an exploit on building blocks that have not previously been faulted, even if it only serves as a reminder that not only obvious targets such as the polynomial comparison should be protected.

The generic variable $z \in \mathcal{Z}$ in Fig. 1 is instantiated with an uncompressed message coefficient $m \in [0, \rho-1]$. Although practically any distribution of its faulted counterpart $m^\star$ enables the attack, at least if the distribution is known to the attacker, we again idealize the case where $m^\star$ is uniformly distributed on $[0, \rho-1]$. Leveraging fault propagation, the attack surface consists of $\mathsf{Decompress}(v; \rho, \delta_v)$, a butterfly in the final layer of the INTT, and a modular subtraction. Recall that the modular subtraction is balanced according to Lemma 3, i.e., a uniformly distributed fault in the butterfly or decompression output results in a uniformly distributed $m^\star$. Given that primes ρ are odd, the final decryption step, i.e., $m^\star \leftarrow \mathsf{Compress}(m^\star; \rho, 1)$ as defined in Eq. (5), is inherently biased. As illustrated in Fig. 5 for $\rho = 7$, the compression function maps $\lfloor \rho/2 \rfloor = 3$ coefficients in $[0, \rho-1]$ to $m^\star = 0$, whereas $\lceil \rho/2 \rceil = 4$ coefficients map to $m^\star = 1$.

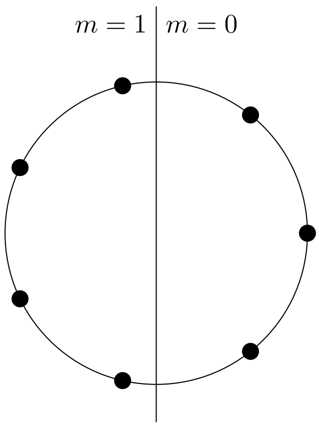

Figure 5: Message coefficients m before and after compression according to Eq. (5) where prime $\rho = 7$.

For the actual prime $\rho = 3329$ used in KYBER, the right and left semicircles contain $\lfloor \rho/2 \rfloor = 1664$ and $\lceil \rho/2 \rceil = 1665$ field elements respectively. Hence, the probability of a failed decapsulation is $1665/3329 \approx 50.015\%$ if the original message bit $m = 0$ and $1664/3329 \approx 49.985\%$ otherwise. At least in theory, a measurement of this failure rate suffices to recover m. For $n = 18201189$ perfectly faulted decapsulations, the recovery succeeds with 90% certainty, as can be derived from the *cumulative distribution function* (CDF) of a binomial distribution: $F_{\text{bino}}(\lfloor n/2 \rfloor; n, 1664/3329) \geq 90\%$ where n is odd. Apart from the staggering number of faults, the attack is hampered in practice because fault injections are unlikely to be perfect, and the probability that no fault is injected is typically unknown.

C Proofs

C.1 Lemma 2

The case $\mathsf{F} : \mathcal{A} \rightarrow \mathcal{C}$ of Lemma 2 is proven in Eq. (21); the case $\mathsf{F} : \mathcal{A} \times \mathcal{B} \rightarrow \mathcal{C}$ is proven in Eq. (22).

$$\Pr(c) = \sum_{\substack{a \in \mathcal{A} \text{ s.t.} \\ \mathsf{F}(a) = c}} \Pr(a) = \frac{|\mathcal{A}|}{|\mathcal{C}|} \cdot \frac{1}{|\mathcal{A}|} = \frac{1}{|\mathcal{C}|}. \tag{21}$$

$$\begin{aligned}\Pr(c) &= \sum_{\substack{(a,b)\in\mathcal{A}\times\mathcal{B}\\ \text{s.t. } \mathsf{F}(a,b)=c}} \Pr(a \wedge b) = \sum_{b\in\mathcal{B}} \Pr(b) \sum_{\substack{a\in\mathcal{A} \text{ s.t.}\\ \mathsf{F}(a,b)=c}} \Pr(a) \\ &= \frac{|\mathcal{A}|}{|\mathcal{C}|} \cdot \frac{1}{|\mathcal{A}|} \cdot \sum_{b\in\mathcal{B}} \Pr(b) = \frac{1}{|\mathcal{C}|}.\end{aligned} \tag{22}$$

C.2 Lemma 3

Balancedness with respect to input $a \in \mathcal{R}$ in Lemma 3 is proven in Eq. (23) and follows from the property that each element in a ring has an *additive inverse*. Balancedness with respect to input $b \in \mathcal{R}$ is proven in an identical manner.

$$\forall (b,c) \in \mathcal{R}^2, |\{a \in \mathcal{R} \mid a + b = c\}| = |\{c - b\}| = 1. \tag{23}$$

C.3 Lemma 4

Balancedness with respect to input $a \in \mathcal{F}$ in Lemma 4 is proven in Eq. (24) and follows from the property that each element $b \neq 0$ in a field has an *multiplicative inverse*.

$$\forall (b,c) \in \mathcal{F}^2, |\{a \in \mathcal{F} \mid a \cdot b = c\}| = |\{c \cdot b^{-1}\}| = 1. \tag{24}$$