Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber

Zhuang Xu[1,2,3], Owen Pemberton[3], Sujoy Sinha Roy[3], David Oswald[3]

[1] School of Mathematical Sciences, Shenyuan Honors College and BDBC, Beihang University, and LMIB, Ministry of Education, Beijing 100191, China
[2] Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China
xu_zhuang@buaa.edu.cn
[3] School of Computer Science, University of Birmingham, United Kingdom
o.m.pemberton@pgr.bham.ac.uk, s.sinharoy@cs.bham.ac.uk, d.f.oswald@bham.ac.uk

Abstract. In this paper, we propose EM side-channel attacks with carefully constructed ciphertext on Kyber, a lattice-based key encapsulation mechanism, which is a candidate of NIST Post-Quantum Cryptography standardization project. We demonstrate that specially chosen ciphertexts allow an adversary to modulate the leakage of a target device and enable full key extraction with a small number of traces through simple power analysis. Compared to prior research, our techniques require a lower number of traces and avoid the need for template attacks. We practically evaluate our methods using both a clean reference implementation of Kyber and the ARM-optimized pqm4 library. For the reference implementation, we target the leakage of the output of the inverse NTT computation and recover the full key with only four traces. For the pqm4 implementation, we develop a message-recovery attack that leads to extraction of the full secret-key with between eight and 960 traces (or 184 traces for recovering 98% of the secret-key), depending on the compiler optimization level. We discuss the relevance of our findings to other lattice-based schemes and explore potential countermeasures.

Keywords: Lattice-based cryptography · Kyber · Side-channel analysis · Chosen-ciphertext attack

1 Introduction

Invented in 1970s, public-key cryptography made it possible to establish a shared secret-key between two parties communicating over a public channel without using a prior shared secret. Since then, public-key cryptography research has resulted in public-key encryption, key agreement and digital signature schemes. Internet standards such as TLS, S/MIME and PGP use public-key cryptography at their core. Even today's state of the art privacy preserving computation techniques such as homomorphic and functional encryption are augmentations of public-key encryption.

The most widely used public-key cryptographic algorithms are the RSA [RSA78] and Elliptic Curve cryptosystems [Mil85]. The security of these two cryptosystems relies on the hardness of the integer factorization and discrete logarithm problems. These two problems are *presumed* to be computationally infeasible to solve using traditional computers. However, Shor's algorithm [Sho97], a quantum algorithm, can solve both integer factorization and discrete logarithm in polynomial-time and thus break the RSA and Elliptic Curve cryptosystems efficiently using a large-scale quantum computer. Rapid

advancement in quantum computing in the recent years have made the development of small-scale quantum computers possible [AAB+19]. Many quantum computing scientists and cryptographers anticipate that large-scale quantum computers able to break RSA and Elliptic Curve cryptosystems will be feasible within the next 20 years [Nat20]. If such quantum computers are ever built, they will completely destroy the present-day public-key infrastructure. Hence, we need next-generation public-key cryptographic algorithms that cannot be broken by quantum computers.

Post Quantum Cryptography (PQC) refers to the design and analysis of cryptographic algorithms based on computational problems that are presumed to be unsolvable using both traditional and quantum computers. Early post-quantum algorithms, such as the code-based McEliece public-key encryption and hash-based Merkle signature, date back to 1970s. The development of new-generation post-quantum public-key algorithms has gained significant attention since around 2010 and as a consequence several efficient post-quantum public-key encryption, key agreement and digital signature schemes have been proposed. To standardize post-quantum public-key algorithms, NIST initiated the PQC standardization project in 2017 and called for proposals. Depending on the underlying mathematical problem, PQC schemes can be classified into five categories: code-based, hash-based, lattice-based, multivariate-based and super-singular isogeny-based. Since the beginning of the PQC standardization project, the candidates have been analyzed for their resistance against mathematical cryptanalysis. On completion of the first round of the standardization project in January 2019, a total of 26 candidates have moved to the second round. It is anticipated that the third round of the standardization project is likely to start in the second half of 2020.

Ongoing mathematical cryptanalysis efforts by the cryptography research community have strengthened confidence in the security of existing PQC candidates and such efforts are likely to continue for the next several years. As post-quantum cryptography will be deployed on a wide range of computing platforms, it is extremely important that the candidate algorithms are also scrutinized for their resistance against physical cryptanalysis. In this paper we analyze a lattice-based Key Encapsulation Mechanism (KEM) algorithm Kyber [BDK+18] and propose practical and efficient side-channel attacks to recover the long-term secret-key.

Related Work There have been several Chosen-Ciphertext Attacks (CCA) [Flu16, DCQ19, QCD19, BDHD+19, BGRR19] on lattice-based public-key encryption schemes that are considered secure in the Chosen-Plaintext Attack model, *i.e.*, IND-CPA. These attacks aim to recover a long-term secret-key by exploiting the presence of a plaintext checking oracle. The CCA attacker sends malformed ciphertexts to a decryption algorithm and queries the plaintext checking oracle to learn if the guesses about the decrypted messages are correct. In this attack, the ciphertexts are crafted in such a way that the responses generated by the plaintext checking oracle can help the attacker discern the long-term secret-key. Note that the attacker can only discover the secret-key if it is reused or long-term as it is necessary to have multiple interactions with the decryption algorithm and plaintext checking oracle.

Lattice-based public-key schemes can be made secure against CCA by being transformed into IND-CCA schemes with the help of a post-quantum variant of the Fujisaki-Okamoto (FO) transformation [FO99]. The FO transformation performs a re-encryption of the decrypted message and compares the computed ciphertext with the received ciphertext. Due to this ciphertext matching, any malformed ciphertext gets detected by the decryption algorithm and thereafter the decrypted message is not outputted, thus disabling the existence of a theoretical plaintext checking oracle. All lattice-based public-key schemes that have proceeded to the second round of NIST's PQC standardization project apply FO transformation (thus IND-CCA) and hence they are not vulnerable to *mathematical*

long-term secret-key recovery attacks of the above type.

Although the existing lattice-based public-key schemes are resistant against known mathematical cryptanalytic attacks, their security against potential physical cryptanalytic attacks is yet to be studied in depth. Realistically, post-quantum schemes will be used in many applications where the biggest threats will appear from invasive or noninvasive side-channel and fault attackers. For example, D'Anvers et al. [DTVV19] demonstrated a long-term key-recovery attack on the IND-CCA LAC [LLZ+18] public-key encryption scheme by exploiting timing leakage from its nonconstant-time decapsulation algorithm. In their attack, carefully constructed ciphertexts are sent to the decryption block of LAC such that, depending on the value of a targeted secret-coefficient, a nonconstant-time error correction code is called *conditionally* to rectify any error in the decrypted message. Although LAC is resistant against mathematical chosen-ciphertext attacks, the timing leakage from the decryption opens a plaintext-checking oracle to an attacker. Ravi et al. [RRCB20] demonstrated a more powerful generic Electro-Magnetic (EM) side-channel assisted long-term key recovery attack technique on several lattice-based public-key encryption and KEMs (constant-time) that have proceeded to the second round of the standardization project. However, a large number of traces are still needed for some KEMs, e.g., full key recovery from Kyber512 needs around 7,680 traces. This motivated us to investigate more efficient and practical side-channel assisted key-recovery attacks targeting harder-to-break lattice-based schemes, such as Kyber.

Other side-channel attacks on lattice-based public-key cryptosystems include template attack by Primas et al. [PPM17] and horizontal differential power analysis (DPA) attack by Aysu et al. [ATT+18]. Recently Huang et al. [HCY20] applied several power analysis methods to recover the private key from NTRU-Prime [BCLvV17]. Most of these attacks target primitive computations, such as polynomial multiplications, where the long-term secret-key is used as an operand.

Another set of side-channel assisted attacks target message recovery from lattice-based public-key encryption or key encapsulation schemes. Such message-recovery attacks could potentially be used to perform session-key recovery in TLS. Amiet et al. [ACLZ20] demonstrated a message recovery attack based on leakage from the encoding function of a reference implementation of NewHope [ADPS16]. Ravi et al. [RBRC20] proposed methods to perform message recovery from vulnerabilities in the message decoding procedure of several lattice-based schemes. Their attack can be turned into a single-trace message recovery attack if a large number of traces are available to build a template base. All these physical attacks show that there are many vulnerabilities in these next-generation lattice-based public-key schemes, and furthermore they give a strong indication that many more potential vulnerabilities are yet to be discovered. Hence, we believe that further study is essential in this area to discover more vulnerabilities such that necessary countermeasures can be developed to make lattice-based post-quantum public-key cryptography secure for real-world deployment.

Despite many potential targets within lattice-based KEM shown in previous sudies, how to reveal the secret-dependent informatin efficiently is less studied. Possible problems based on the unique structure of Kyber, one of the candidates in Round 2, remain to be investigated comprehensively and thoroughly from more perspectives although there have been some Side-Channel Analysis (SCA) works about its analysis [RRCB20, ACLZ20, RBRC20].

Contributions In this paper we present a practical side-channel assisted Chosen-Ciphertext Attack on IND-CCA Kyber KEM and show how to recover the long-term secret-key using a small number of traces (between four and 960, depending on the target implementation and specific technique). We focus on Simple Power Analysis (SPA) [KJJ99]-like methods that avoid the need for constructing templates, thus preventing the associated portability

issues [EG12]. In particular, we propose the following new attack techniques:

1. We show how to construct special ciphertexts such that we can modulate side-channel leakage from the device under test and *efficiently* classify a targeted secret coefficient using a minimum number of traces.
2. To experimentally validate the attack techniques, first we target the `pqm4` reference C implementation of Kyber. We observe that several functions leak secret-dependent information through EM emanation. Using only four specially crafted ciphertexts, we can recover the entire secret-key through SPA.
3. We then target the ARM-optimized implementation of Kyber from the `pqm4` library, which does not expose the same strong leakage as the reference implementation. However, we find that the decrypted message can be recovered from a small number of traces with SPA. We show how message recovery, together with specifically chosen ciphertexts, can be used to fully recover the secret-key. Depending on the compiler optimization level, this attack can recover the full key with between 8 (`-O0`) and 960 traces (`-O3`).

Finally, we also discuss countermeasures, including those based on discarding ciphertexts with a special structure possibly indicating an attack.

All the experiments for the reference implementation of Kyber used the `pqm4` library commit `c32bcd0`. All experiments for the ARM-optimized library used commit `84c5f91`. We provide our data sets and code under the following link:
https://mega.nz/folder/YBcRyAhQ#FDC9wpc6siYQg-eMxHLJhA

Organization of the Paper The remainder of this paper is organized as follows. In Section 2, we introduce our notation and the mathematical background. In Section 3, we present our chosen-ciphertext SPA attack on a "clean" implementation of Kyber, while in Section 4 we focus on attacks on highly optimized ARM implementations. We conclude in Section 5, discussing potential countermeasures.

2 Preliminaries

In this section we briefly describe the mathematical background and define notations necessary to understand this paper.

2.1 Notations

Let q be a prime and $\mathbb{Z}_q$ be the ring of integers modulo q. Centered modular reduction is represented as $r' = r \mod^{\pm} q$ where $r' \in \left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$. Whereas, the other modular reduction $r' = r \mod^{+} q$ means $r' \in [0, q-1]$.

We define the ring of polynomials $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$, for some integer N, typically a power of two. Thus the polynomials have N coefficients where each coefficient is modulo q. Similarly, $\mathcal{R}_q^{l\times k}$ denotes the ring of $l \times k$-matrices over any ring $\mathcal{R}_q$. One-dimensional matrix is essentially called a vector. The transpose of a vector or a matrix is represented using the superscript ^{T}. A single polynomial is represented using a normal-font lowercase letter (such as a), a vector of polynomials is represented using a bold-font lowercase letter (such as $\mathbf{a}$), and a matrix of polynomials is represented using a bold-font capital letter (such as $\mathbf{A}$). Multiplication in any ring is denoted using $\cdot$ operator whereas point-wise multiplication is denoted using $\circ$ operator. For a polynomial a, the ith coefficient is represented as $a[i]$, and the ith component of a vector is denoted as $\mathbf{a}_i$. The Number Theoretic Transform (NTT) of any element a is represented as $\hat{a}$. For an element $x \in \mathbb{Q}$, we use $\lceil x \rfloor$ to present the closest integer to x; the fractional part of x is rounded up. The uniform distribution is denoted as $\mathcal{U}$, and a centered binomial distribution with parameter

μ is denoted as β_μ. We use the notations $a \leftarrow \mathcal{U}$ and $a \leftarrow \beta_\mu$ to denote that a is randomly sampled from $\mathcal{U}$ or β_μ respectively. For side-channel measurements ("traces"), we refer to a single trace i as $p_i\,(t)$, where t is discretized time in sample points.

2.2 Module Learning with Errors Problem

The Learning with Errors (LWE) problem, introduced by Regev [Reg05] in 2005, is a mathematical problem that has been used as a foundation for a large number of lattice-based cryptosystems. The LWE problem is presumed to be computationally infeasible to solve for both classical and quantum computers. An LWE distribution consists of tuples of the form

$$\left(\mathbf{a}, b = \mathbf{a}^T\mathbf{s} + e\right) \in \mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_q\,, \tag{1}$$

where the secret vector $\mathbf{s} \leftarrow \beta_\mu(\mathbb{Z}_q^{l\times 1})$ is fixed for all samples, uniformly random $\mathbf{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1})$ is fresh, and error $e \leftarrow \beta_\mu(\mathbb{Z}_q)$ is also fresh.

The decisional LWE problem states that it is computationally infeasible to distinguish uniformly random samples $(\mathbf{a}, u) \leftarrow \mathcal{U}(\mathbb{Z}_q^{l\times 1} \times \mathbb{Z}_q)$ from the LWE samples in Equation (1). The search LWE problem asks to compute the secret $\mathbf{s}$ given several LWE samples.

A module version of the LWE problem, known as the Module Learning with Errors (MLWE), uses the polynomial ring $\mathcal{R}_q$ instead of the integer ring $\mathbb{Z}_q$. Thus, an MLWE distribution consists of tuples of the form

$$\left(\mathbf{a}, b = \mathbf{a}^T\mathbf{s} + e\right) \in \mathcal{R}_q^{l\times 1} \times \mathcal{R}_q\,, \tag{2}$$

where $\mathbf{a}$ is a vector of randomly generated polynomials in $\mathcal{R}_q$, $\mathbf{s}$ is a secret vector of polynomials in $\mathcal{R}_q$ of which the coefficients are sampled from β_μ, and e is an error polynomial whose coefficients are sampled from β_μ. The decision and search LWE problem over standard lattices can be extended to the decision and search MLWE problem over module lattices. Module lattices have been used to construct flexible and secure public-key encryption and KEMs such as Kyber [SAB+19], Saber [DKSRV18] and a signature scheme Dilithium [DLL+18]. Since we propose side-channel attacks on Kyber, in the following subsection, we briefly describe the Kyber KEM algorithm.

2.3 Kyber

Kyber [BDK+18] is an MLWE-based IND-CCA KEM that has proceeded to the second round of the NIST's post-quantum cryptography standardization project. It comes with three variants namely Kyber512, Kyber768 and Kyber1024 targeting security levels similar to AES-128, AES-192 and AES-256 respectively. The variants perform arithmetic operations in a fixed polynomial ring $\mathcal{R}_q = \mathbb{Z}_q/(x^{256}+1)$ where $q = 3329$ is a prime; they use the module dimensions 2, 3, and 4 to achieve the three security levels. Additionally, the variants use the same binomial distributions with parameter $\mu = 2$. In this section we briefly describe the algorithms that are at the core of Kyber. For full description of Kyber, readers are referred to the original specification of Kyber [SAB+19].

The IND-CCA Kyber KEM is based on the IND-CPA Kyber Public Key Encryption (PKE) with the application of a post-quantum variant of the FO transformation [FO99]. Simplified versions of the IND-CPA encryption and decryption algorithms are presented in Algorithm 1 and Algorithm 2 respectively. The encryption operation takes a public-key pk, a binary message m, and a random coin r as inputs. The public-key is concatenation of a random seed $seed_{\mathbf{A}}$, and a vector $\hat{\mathbf{t}}$ of polynomials. In line 1 of Algorithm 1, the random seed is expanded using an Extendable Output Function (XOF) to generate the matrix $\hat{\mathbf{A}}$ directly in the NTT domain. Next, two vectors of polynomials $\mathbf{r}$ and $\mathbf{e_1}$ are sampled from a centered binomial distribution β_μ, and in line 5 the NTT of $\mathbf{r}$ is computed. Since

both $\hat{\mathbf{A}}^T$ and $\mathbf{r}$ are in the NTT domain, the matrix-vector multiplication in line 6 is a coefficient-wise operation. Thereafter, an inverse-NTT is required before adding the error vector $\mathbf{e_1}$ to the result of matrix-vector multiplication. In line 7, the binary message string is encoded into a message polynomial and then added to the result of polynomial-vector multiplication $\mathbf{t} \cdot \mathbf{r}$ (efficiently computed using NTT) along with an error polynomial e_2. Finally, the ciphertext consists of two components c_1 and c_2 that are derived from $\mathbf{u}$ and v.

The IND-CPA decryption in Algorithm 2 receives a ciphertext $c = (c_1 \parallel c_2)$ and the secret-key. In line 1 and 2, $\mathbf{u}$ and v are computed c_1 and c_2. In line 4, the secret polynomial-vector is multiplied by the ciphertext polynomial-vector $\mathbf{u}$ and the result is eventually brought back to the time-domain by performing inverse-NTT. Finally, the message, which is a binary string, is recovered by performing coefficient-wise compression and encoding. The IND-CPA encryption and decryption algorithms are developed into IND-CCA encapsulation and decapsulation with the application of a post-quantum variant of the FO transformation in Algorithm 3 and Algorithm 4 respectively. The FO transformation steps use two hash functions G and H and a Key Derivation Function (KDF). The encapsulation returns an IND-CCA ciphertext c and a session key K.

Algorithm 1 Simplified Kyber.CPAPKE.Enc $\left(pk = (seed_{\mathbf{A}} \parallel \hat{\mathbf{t}}), m, r\right)$ [SAB+19]

1: $\hat{\mathbf{A}}^T := \text{XOF}(seed_{\mathbf{A}}) \in \mathcal{R}_q^{k \times k}$ /* Here k is the module-dimension */
2: $\mathbf{r} := \beta_\mu(\mathcal{R}_q^{k \times 1}; r)$
3: $\mathbf{e}_1 := \beta_\mu(\mathcal{R}_q^{k \times 1}; r)$
4: $e_2 := \beta_\mu(\mathcal{R}_q; r)$
5: $\hat{\mathbf{r}} := \text{NTT}\left(\mathbf{r}\right) \in \mathcal{R}_q^{k \times 1}$
6: $\mathbf{u} := \text{NTT}^{-1}\left(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}\right) + \mathbf{e}_1 \in \mathcal{R}_q^{k \times 1}$
7: $v := \text{NTT}^{-1}\left(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}\right) + e_2 + \text{Decompress}_q\left(\text{Decode}_1\left(m\right), 1\right) \in \mathcal{R}_q$
8: $c_1 := \text{Encode}_{d_u}\left(\text{Compress}_q\left(\mathbf{u}, d_u\right)\right)$
9: $c_2 := \text{Encode}_{d_v}\left(\text{Compress}_q\left(v, d_v\right)\right)$
10: **return** $c = (c_1 \parallel c_2)$

Algorithm 2 Simplified Kyber.CPAPKE.Dec $\left(sk, c = (c_1 \parallel c_2)\right)$ [SAB+19]

1: $\mathbf{u} := \text{Decompress}_q\left(\text{Decode}_{d_u}\left(c_1\right)\right)$
2: $v := \text{Decompress}_q\left(\text{Decode}_{d_v}\left(c_2\right)\right)$
3: $\hat{\mathbf{s}} := \text{Decode}\left(sk\right)$
4: $m := \text{Encode}_1\left(\text{Compress}_q\left(v - \text{NTT}^{-1}\left(\hat{\mathbf{s}}^T \circ \text{NTT}\left(\mathbf{u}\right)\right), 1\right)\right)$
$\triangleright\, m := \text{Compress}_q\left(v - \mathbf{s}^T \cdot \mathbf{u}, 1\right)$
5: **return** m

The decapsulation Algorithm 4 takes a ciphertext c and the KEM secret-key sk_{KEM} ($sk_{KEM} := (sk \parallel pk \parallel \text{H}(pk) \parallel z)$) as inputs. The IND-CPA decryption is used to obtain the decrypted message m'. Next, the decrypted message is re-encrypted using the public-key pk and the re-encrypted ciphertext is c'. Now, the received ciphertext c and re-encrypted ciphertext c' are compared to detect Chosen-Ciphertext Attack (CCA). If the received ciphertext is genuine, then a session key K is returned from the KDF. Otherwise, a pseudorandom string is returned.

Our attack target is the core long-term secret-key $\mathbf{s}$ in the decryption phase (Alg. 2) during decapsulation (Alg. 4). Note that the coefficients of polynomials in $\mathbf{s}$ are sampled from a binomial distribution and their values range from -2 to 2 only.

Algorithm 3 Simplified Kyber.CCAKEM.Enc (pk)

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$m \leftarrow \mathrm{H}(m)$
$\left(\bar{K}, r\right) := \mathrm{G}(m \parallel \mathrm{H}(pk))$
$c :=$ Kyber.CPAPKE.Enc (pk, m, r)
$K := \mathrm{KDF}\left(\bar{K} \parallel \mathrm{H}(c)\right)$
return (c, K)

Algorithm 4 Simplified Kyber.CCAKEM.Dec (c, sk_{KEM}))

$m' :=$ Kyber.CPAPKE.Dec (sk, c)
$\left(\bar{K}', r'\right) := \mathrm{G}(m' \parallel \mathrm{H}(pk))$
$c' :=$ Kyber.CPAPKE.Enc (pk, m', r')
if $c = c'$ **then**
  return $K := \mathrm{KDF}\left(\bar{K}' \parallel \mathrm{H}(c)\right)$
else
  return $K := \mathrm{KDF}(z \parallel \mathrm{H}(c))$ /* Here z is a pseudorandom string */
end if

2.4 Experimental Setup

For all subsequent experiments, we compiled and ran the `pqm4` target implementations on an STM32F407G discovery development board [STM16]. We use a PicoScope 6404C digital oscilloscope with a Langer RF-U 5-2 H-field probe to record EM traces at a fixed position over the Microcontroller (µC). A ZFL-1000LN+ low-noise amplifier between the probe and oscilloscope amplifies the signal by 20 dB. Our power traces are originally sampled at 2.5 GHz, then digitally downsampled to 500 MHz before analysis. We used the original implementations of `pqm4` [KRSS], with the only addition being a trigger to simplify the recording of traces. The STM32 chip runs at a clock frequency of 168 MHz.

Adversary Model For all the attacks, we assume an adversary who can collect a number of side-channel traces during the KEM decapsulation (Algorithm 2). We assume a chosen-ciphertext scenario, where the adversary can repeatedly request decapsulation of arbitrary ciphertext c.

3 Simple Power Analysis of `pqm4` Kyber Reference Implementation

In this section, we analyze the `pqm4` "clean" implementation [KRSS] of Kyber from a side-channel perspective. More precisely, recent versions of `pqm4` include the `PQClean` library as a submodule that provides independent and portable C implementations of the supported algorithms. We show that we can mount an SPA attack using few traces for successfully recovery of the long-term secret-key. For our initial analysis, we run the Kyber512 KEM on the STM32F407G and collect EM traces when the decryption is called. We set the coefficients of the first half of the secret-key s_0 (where $\mathbf{s} = (s_0, s_1)$ for Kyber512) as follows:

$$s_0[i] = \begin{cases} -2, & \text{for } i = 0, 1, ..., 49; \\ -1, & \text{for } i = 50, 51, ..., 99; \\ 0, & \text{for } i = 100, 101, ..., 155; \\ 1, & \text{for } i = 156, 157, ..., 205; \\ 2, & \text{for } i = 206, 207, ..., 255. \end{cases}$$

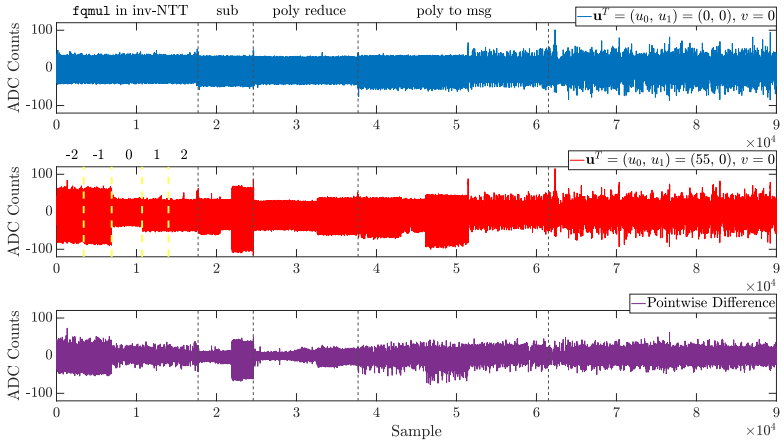

Figure 1: EM traces of Kyber-CPAPKE decryption on the STM32F407G. Top, blue: $\mathbf{u}^T = (u_0, u_1) = (0, 0)$, $v = 0$; middle, red: $\mathbf{u}^T = (u_0, u_1) = (55, 0)$, $v = 0$; bottom, purple: difference between top and middle trace.

Figure 1 shows traces of the final part of decryption, beginning with the final step of the inverse NTT, for two choices of $\mathbf{u}$ and v. This initial test shows that several functions in the decryption leak the secret-key coefficients: in the middle trace in Figure 1, one can easily visually distinguish *classes* of the coefficient values ($\{-2, -1\}$, 0 and $\{1, 2\}$). In the following, we analyze the relevant functions in detail and show how a chosen-ciphertext SPA attack can be constructed to exploit this leakage and recover the full secret-key.

3.1 Simple Power Analysis of Modular Reduction in Inverse NTT

In this section, we focus on the output of the inverse NTT in Kyber, *i.e.*, Line 4 in Algorithm 2:

$$\text{NTT}^{-1}\left(\text{NTT}\left(\mathbf{s}^T\right) \circ \text{NTT}\left(\mathbf{u}\right)\right)$$

where $\mathbf{s}^T = (s_0, s_1)$ and $\mathbf{u}^T = (u_0, u_1)$ for Kyber512 (*i.e.*, module dimension 2). The final step of inverse NTT performs coefficient-wise modular multiplications with a constant (`CLEAN_KYBER512_CLEAN_zetas_inv[127]` = 1441). For example, the clean C implementation of Kyber performs integer multiplication[1] followed by Montgomery reduction[2] (mod $^{\pm}q$) using the `fqmul()` function as shown in Listing 1.

```
1 void PQCLEAN_KYBER512_CLEAN_invntt(int16_t poly[256]) {
2     ...
3     for (j = 0; j < 256; ++j) {
4         poly[j] = fqmul(poly[j], PQCLEAN_KYBER512_CLEAN_zetas_inv[127]);
5     }
6 }
```

Listing 1: Final step of NTT in clean C implementation

[1] https://github.com/PQClean/PQClean/blob/8db3/crypto_kem/kyber512/clean/ntt.c#L116
[2] https://github.com/PQClean/PQClean/blob/8db3/crypto_kem/kyber512/clean/reduce.c#L17

Next, we consider the part of the trace belonging to the execution of inverse NTT, *i.e.*, the first region in Figure 1. First, we determine the Points of Interest (PoI) for further analysis. Based on our profiling, we select downward peaks (cf. Figure 2) in the trace as PoI as we observe significant variance for different secret coefficient values for this particular choice. As `fqmul()` is invoked coefficient-by-coefficient, each of such peaks corresponds to one coefficient of the secret-key.

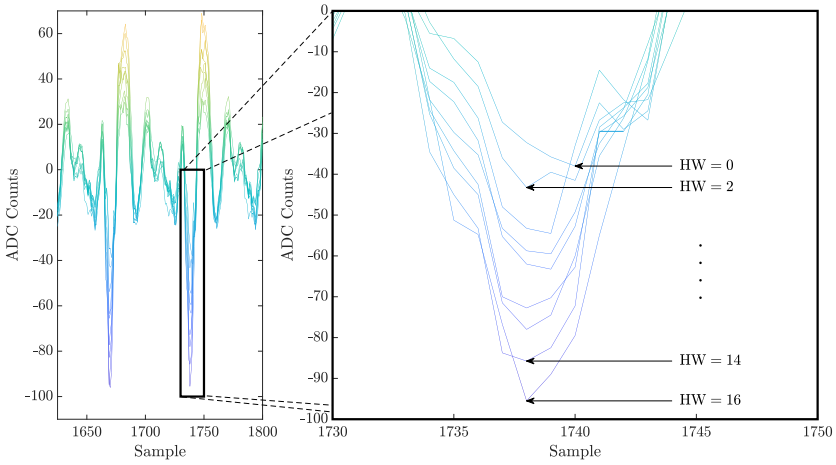

Figure 2: Leakage for different output Hamming Weights (HWs) of `fqmul()` at PoI

Leakage Model We find that the trace at PoI_i is approximately proportional to the HW of the output of the ith invocation of `fqmul()`, as shown in the following standard HW leakage model:

$$|p(\text{PoI}_i)| \approx a \times \text{HW}\left(\left(\mathbf{s}^T \cdot \mathbf{u} \mod^{\pm} q\right)[i]\right) + \mathcal{N}$$

Where a is a scaling factor and $\mathcal{N}$ is a Gaussian noise term.

A Preliminary Idea Whilst `fqmul()` employs a constant-time Montgomery reduction, the EM trace allows an adversary to exploit the data dependency. However, as evident from Figure 1, the leakage is only visible upon certain choices of $\mathbf{u}$ and v. Hence, the adversary has to craft appropriate such values to make the secret-key "visible" to SPA. In the following, we explore in detail how such values can be constructed.

From Figure 1 with $\mathbf{u}^T = (u_0, u_1) = (55, 0)$, we can distinguish between the classes $\{-2, -1\}$, 0 and $\{1, 2\}$. However, that does not reveal the whole key, e.g., coefficients of -2 and -1 cannot be distinguished. A first intuition is to choose $\mathbf{u}^T = (u_0, u_1) = (1, 0)$. Because `fqmul()` computes $\mathbf{s}^T \circ \mathbf{u} \mod^{\pm} q = (s_0 \cdot u_0 + s_1 \cdot u_1) \mod^{\pm} q$, the output is in this case equal to s_0. This would directly reveal half of the secret-key through side-channel leakage. However, this intuition of setting the polynomials of $\mathbf{u}$ to any chosen constant is not valid in practice as the derivation of $\mathbf{u}$ from a ciphertext follows a special structure that allows the coefficients to have values in a certain range (explained in the following section). Hence, we need a better strategy for crafting the chosen ciphertexts.

Constructing Chosen Ciphertext First, note that $\mathbf{u}$ and v are never exchanged directly; during encryption they are compressed and then encoded (Algorithm 1) as ciphertext $c = (c_1 \parallel c_2)$. During decryption (Algorithm 2), $\mathbf{u}$ and v are computed by decompressing the received ciphertext $c = (c_1 \parallel c_2)$. Since the compression causes data loss, the

decompression of ciphertext produces approximate of $\mathbf{u}$ and v only. In detail, the coefficient-wise compression and decompression functions in Kyber work as follows.

$$\mathbf{u}_{\text{compressed}} = \text{Compress}_q\left(\mathbf{u},\, d_u\right) = \left\lceil \frac{2^{d_u}}{q}\mathbf{u} \right\rfloor \mod^+ 2^{d_u} = \left\lceil \frac{2^{10}}{3329}\mathbf{u} \right\rfloor \mod^+ 2^{10}$$
$$\mathbf{u}' = \text{Decompress}_q\left(\mathbf{u}_{\text{compressed}},\, d_u\right) = \left\lceil \frac{q}{2^{d_u}}\mathbf{u}_{\text{compressed}} \right\rfloor = \left\lceil \frac{3329}{2^{10}}\mathbf{u}_{\text{compressed}} \right\rfloor$$

Due to data loss, the $\mathbf{u}$ from encryption (Algorithm 1) might produce a different $\mathbf{u}'$ in decryption (Algorithm 2). Furthermore, one can see that the coefficients of the decompressed polynomials in $\mathbf{u}'$ cannot have a value 1. Only for a subset $U = \left\{ \left\lceil \frac{3329}{2^{10}} \cdot i \right\rfloor : i = 0,\, 1,\, \ldots,\, 1023 \right\}$ of $\mathbb{Z}_q$, we can have a bijection between $\mathbf{u}$ and $\mathbf{u}'$. Similarly, for v in Algorithm 1 and v' in Algorithm 2, bijection works in the subset $V = \left\{ \left\lceil \frac{3329}{2^3} \cdot i \right\rfloor : i = 0,\, 1,\, \ldots,\, 7 \right\}$ only. Hence, we create the malicious ciphertexts with coefficients from U and V only.

Selecting Appropriate Chosen Ciphertexts After ensuring that the chosen $\mathbf{u}$ and v properly "propagate" through compression and decompression, we need to select special values for u_0 and u_1 (where $\mathbf{u}^T = (u_0, u_1)$ in Kyber512) with coefficients from U to distinguish different coefficients in the secret-key. In the following we discuss recovering the first secret-key polynomial s_0 from $\mathbf{s}$ only as the recovery of the second polynomial s_1 will be a straightforward repetition of the attack steps. We set u_1 to 0 to remove the influence of s_1 on the output of `fqmul()`. In addition, we only keep the constant term of u_0 so that the output value is only determined by the corresponding key coefficient value.

Let us denote the constant-coefficient of u_0 by $u_0[0]$. We first observe that for some choices, e.g., $u_0[0] = 2324$, we obtain a different HW for each value of a secret coefficient: $\text{HW}\left(-2 \cdot u_0[0]\right) = 11$, $\text{HW}\left(-1 \cdot u_0[0]\right) = 8$, $\text{HW}\left(1 \cdot u_0[0]\right) = 9$, $\text{HW}\left(2 \cdot u_0[0]\right) = 6$, $\text{HW}\left(0\right) = 0$. However, in practice, we analyze noisy leakage and aim for mounting an SPA attack with few traces. Therefore, it is hard to distinguish similar HWs. We experimentally found that for the value $u_0[0] = 2324$, we could only distinguish between zero and nonzero secret values. Hence, instead we opted for selecting a few chosen $u_0[0]$ to build a classifier for recovery of the full secret-key step by step. This approach can either make use of several binary or ternary classifiers, or a combination of the two. To find the *optimal constants*, we first computed $\text{HW}\left(s_{coeff} \cdot u_0[0] \mod^{\pm} q\right)$ for all possible secret coefficient values $s_{coeff} = -2,\, \ldots,\, 2$ and for all possible $u_0[0] \in U$. Figure 3 shows the relation between $u_0[0]$ and $\text{HW}\left(s_{coeff} \cdot u_0[0] \mod^{\pm} q\right)$ under different s_{coeff} values. In this figure, red points denote -2 or -1, blue ones 1 or 2, and black ones 0. It can be seen that there are several candidates for $u_0[0]$ that can distinguish these different coefficients classes.

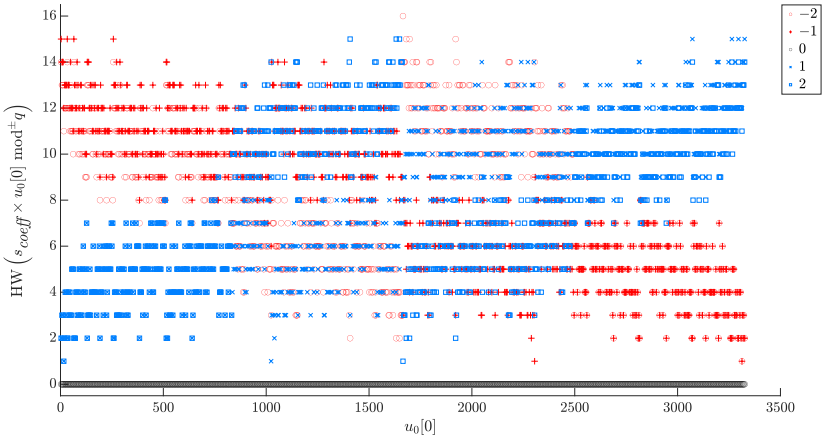

Figure 3: HWs for all possible choices of $u_0[0] \in U$ and s_{coeff} either $\{-2, -1\}$ (blue), 0 (black), or $\{1, 2\}$ (red)

As shown in Figure 3, for small or large $u_0[0]$, one can divide the HWs into two classes: "High" $\{-2, -1\}$ and "Low" $\{0, 1, 2\}$, *i.e.*, admit use of a binary classifier. As $u_0[0]$ grows, we see that there are cases that can divide the HWs into three classes, *i.e.*, admit use of a ternary classifier (with three classes "High", "Medium", "Low"). More precisely:

1. There are 15 possible binary classifiers, *i.e.*, 15 ways to partition a set of 5 values into 2 sets. We only consider classifiers with the following two properties: i) the maximum difference between HWs within a class is ≤ 3; ii). the HW difference between two classes is ≥ 6.
2. There are 40 possible ternary classifiers. We consider those for which: i) the maximum difference between HWs within a class is ≤ 3; ii). the HW difference between two classes is ≥ 4.

In addition, we should keep the HW difference within one class as small as possible and let the HW difference among classes as big as possible to distinguish them. By going over all possible classifiers, we find binary classifiers including the following ones (we use H for "High", L for "Low", and M for "Medium" in the following): $[\mathrm{H} : \{-2, -1\}, \mathrm{L} : \{0, 1, 2\}]$ with $u_0[0] = 3$ and $[\mathrm{H} : \{2, 1\}, \mathrm{L} : \{0, -1, -2\}]$ with $u_0[0] = 3326$. Possible ternary classifiers include $[\mathrm{H} : \{-2, -1\}, \mathrm{M} : \{1, 2\}, \mathrm{L} : 0]$ with $u_0[0] = 55$ and $[\mathrm{H} : \{1, 2\}, \mathrm{M} : \{-1, -2\}, \mathrm{L} : 0]$ with $u_0[0] = 3274$. Using either of those, we can already distinguish $\{-2, -1\}$, 0 and $\{1, 2\}$. It remains to be found a classifier to distinguish -2 from -1 and 1 from 2.

We separately study those cases ($-2/-1$ and $1/2$) as shown in Figure 4. When $u_0[0]$ is < 836 or > 2493, $|\mathrm{HW}(-2 \cdot u_0[0]) - \mathrm{HW}(-1 \cdot u_0[0])| \leq 1$ and $|\mathrm{HW}(2 \cdot u) - \mathrm{HW}(1 \cdot u)| \leq 1$. This difference changes when $u_0[0]$ is between 836 and 2493. It is possible to find ternary classifiers in this region, more precisely: $[\mathrm{H} : \{2, -1\}, \mathrm{M} : \{-1, 2\}, \mathrm{L} : 0]$ with $u_0[0] = 1070$ (HWs 5, 11, 0, 5, 12) and $[\mathrm{H} : \{-2, 1\}, \mathrm{M} : \{1, -2\}, \mathrm{L} : 0]$ with $u_0[0] = 2259$. (HWs 12, 5, 0, 11, 5).

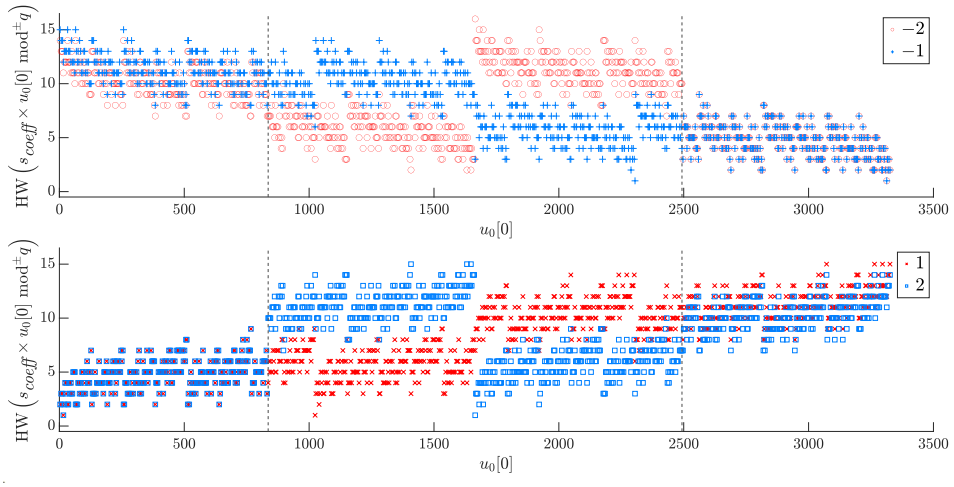

Figure 4: HW difference for $-2/-1$ and $1/2$

Attack Methodology Based on the above classifiers, an SPA attack can be constructed that recovers the secret-key with few traces and without requiring sensitive and detailed profiling as e.g., necessary for template attacks. Concretely, the attack proceeds as follows, using four traces in total:

1. Choose a first $u_0[0] = \text{const}_1 \in U$, for a ternary classifier to distinguish $\{-2, -1\}$, 0 and $\{1, 2\}$ and generate the respective ciphertext c. Record the respective EM trace for `fqmul()` on the target device and, looking at the relevant PoI, determine into which class the respective coefficient falls.
2. Choose a second $u_0[0] = \text{const}_2 \in U$ for a ternary classifier to differentiate $-2/-1$ and $1/2$ and record and evaluate the respective trace.
3. Combining the results from the previous two steps, we recover the coefficients of s_0.

Practical Results We applied our chosen-ciphertext SPA to the C clean implementation of Kyber512 on the STM32F407 (cf. Section 2.4 for details of our setup). We used $\text{const}_1 = 55$ and $\text{const}_2 = 1070$ for the two required ternary classifiers. Figure 5 shows the EM trace for $u_0[0] = 55$ with the PoI classified into the three classes H (-1 or -2), M (1 or 2) and L (0). Similarly, Figure 6 depicts the EM trace for $u_0[0] = 1070$ ($[\text{H} : \{-1, 2\}, \text{M} : \{1, -2\}, \text{L} : 0]$).

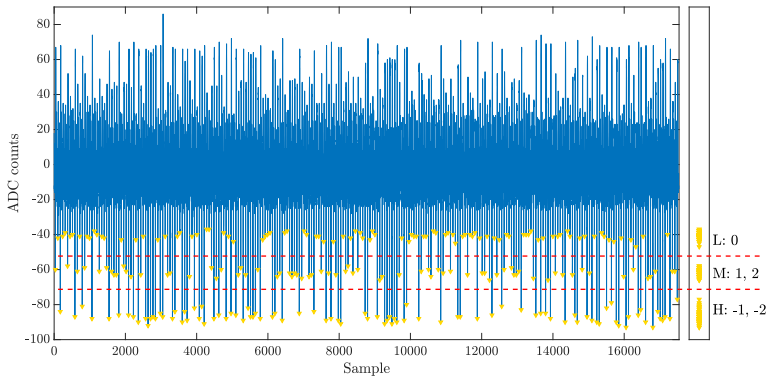

Figure 5: EM trace with classification into three classes for $u_0[0] = 55$

To show the separation between classes, we also project the amplitude at the PoI on

the right. In Figure 5 and Figure 6, the classes are clearly distinguishable, however, we observed other traces with some overlap between the classes. Therefore, in these cases, we used averaging of multiple traces for the same $u_0[0]$. Note that in the ideal case (*i.e.*, success with a single trace without averaging), the full attack requires 4 traces in total (for $u_0[0] = 55$ and $= 1070$ and for $u_1[0] = 55$ and $= 1070$).

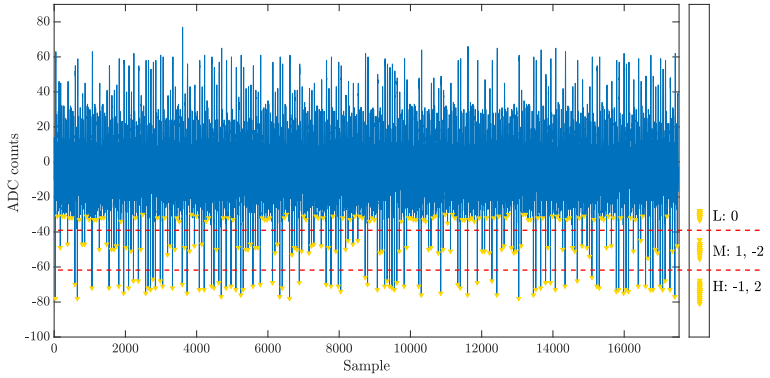

Figure 6: EM trace with classification into three classes for $u_0[0] = 1070$

We executed our attack for 16 randomly chosen secret-keys. Using four traces, we recovered the full key (*i.e.*, all 512 coefficients) in 12 cases, while in three cases 511 of 512 coefficients (*i.e.*, 99.8%) were correctly found. In one case, only 510 of 512 coefficients (*i.e.*, 99.6%) were successfully recovered. However, these remaining one or two coefficients can be found through a trivial exhaustive search over the 5 or 5^2 possible values.

To summarize, the SPA succeeds with four traces in total. In contrast to prior work, due to the careful choice of $\mathbf{u}$, we do not require creating templates, thus avoiding the associated problem of portability between individual devices [EG12].

3.2 Targeting other Functions after the Inverse NTT

As initially shown in Figure 1, subsequent operations after `fqmul()` also exhibit strong leakage that can be "amplified" using chosen ciphertext. While we do not analyze those functions in detail, we note that they can similarly be targeted for key recovery. For example, in the subtraction following the NTT, we have the following leakage model:

$$|p\left(PoI_i\right)| \propto \mathrm{HW}\left(\left(v - \mathbf{s}^T \circ \mathbf{u} \mod^{\pm} q\right)[i]\right)$$

Evidently, in contrast to the attack on `fqmul()`, the other part of the ciphertext (v) also influences the leakage. Hence, an appropriate value for v can be set as:

$$v = \sum_{j=0}^{255} \mathrm{const} \cdot x^j, \ \mathrm{const} \in V$$

Because there are eight possible values for a coefficient of v that maintains the bijection during compression and decompression, we have a larger space of possible chosen ciphertext for v than u_0. Note that as $0 \leq \mathrm{HW}\left(v\right) \leq 6$, a zero coefficient of the secret-key $\mathbf{s}$ no longer automatically falls in the "Low" class. The larger set of possible chosen ciphertext means that there are more possible binary and ternary classifiers, which possibly further facilitates key recovery. Finally, we would like to point out that the technique demonstrated in this section for Kyber in principle can apply to other lattice-based schemes that follow a similar structure. We leave these aspects for future work.

4 Simple Power Analysis of ARM-specific Kyber Implementation in `pqm4`

In the previous Section 3, we showed that a generic, "clean" implementation of Kyber is vulnerable to SPA. In this section, we extend our analysis to the `pqm4` implementations specifically optimized for ARM Cortex M4 µCs. In this optimized version, many functions are manually written in assembly to obtain a memory-efficient, high-speed implementation [BKS19]. As word-parallel additions and subtractions are used and the inverse NTT is re-organized, the attack from Section 3 no longer applies in a straightforward way. We make use of a recent idea to focus on the side-channel leakage of the message m in lattice-based PKE and KEM schemes [ACLZ20, RBRC20]. However, while these two earlier works only considered leaking the message to e.g., obtain the ephemeral shared secret in a KEM, we show that message recovery also allows an adversary to recover the *long-term* secret-key by carefully choosing the ciphertext components $\mathbf{u}$ and v.

To this end, we first focus on the encoding/decoding functions that deal with the messages and discuss their susceptibility to SCA. We then present our side-channel based message recovery for two different optimization levels (`-O0` and `-O3`), and finally show how the long-term secret-key can be recovered from the messages.

4.1 Encoding and Decoding Functions in Decapsulation

We found that there are two operations that are performed coefficient-wise in the ARM-specific implementation, *i.e.*, the encoding and decoding functions named `poly_tomsg()`[3] and `poly_frommsg()`[4] respectively, used in the KEM decapsulation. Algorithm 5 and Algorithm 6 show the pseudocode for encoding and decoding, respectively.

Algorithm 5 Pseudocode of `pqm4` `poly_tomsg()` encoding.

Input: Input polynomial in coeffs[256]

1: **for** $i = 0 \ldots 31$ **do**
2: $\quad \text{msg}[i] = 0;$
3: $\quad$ **for** $j = 0 \ldots 7$ **do**
4: $\quad\quad t = (((\text{coeffs}[8 \cdot i + j] \ll 1) + q/2) \;/\; q) \;\&\; 1;$
5: $\quad\quad \text{msg}[i] \;|= \; t \ll j;$
6: $\quad$ **end for**
7: **end for**
8: **return** msg;

The encoding (Algorithm 5) includes Compress(), which compresses a value from $\mathbb{Z}_q$ into $\mathbb{Z}_2$ (Line 4) and packs 8 bits into one byte in bit-reversed order. Note that in the Kyber KEM, the output of `poly_tomsg()` in the decryption step (Algorithm 2) is the input of decoding in the subsequent re-encryption step (Algorithm 1).

[3] https://github.com/mupq/pqm4/blob/c32b/crypto_kem/kyber768/m4/poly.c#L530
[4] https://github.com/mupq/pqm4/blob/c32b/crypto_kem/kyber768/m4/poly.c#L518

Algorithm 6 Pseudocode of `pqm4` `poly_frommsg()` decoding.

Input: Input message in msg [32]

1: **for** $i = 0 \ldots 31$ **do**
2: **for** $j = 0 \ldots 7$ **do**
3: mask $= -((\text{msg}[i] \gg j)\ \&\ 1)$;
4: coeffs $[8 \cdot i + j] = \text{mask}\ \&\ ((q+1)\ /\ 2)$;
5: **end for**
6: **end for**
7: **return** coeffs [];

Conversely, the decoding function (Algorithm 6) consists of the unpacking of bytes to bits and subsequent decompression, mapping each bit to 0 or $(q+1)\ /\ 2$ (Line 4).

Leakage of Encoding and Decoding Functions From Algorithm 5, we see that $t = 1$ if $(q-1)/4 < \text{coeff}[8 \cdot i + j] < (3q+1)/4$, and otherwise, $t = 0$. The authors of [RBRC20] use the t-test to detect the respective difference in the trace, and then use a template attack to recover m. For Algorithm 6, it is clear that the mask and the output coefficients are different depending on the respective message bit (mask = 0xffff and coeffs $[8 \cdot i + j] = (q+1)/2$ if message bit i is one). The authors of [ACLZ20] targeted a similar function in NewHope, and found that this difference can be observed with one EM trace in a reference implementation without optimization (`-O0`).

In the Kyber KEM, one can choose which function (encode or decode) to target for message recovery. This is because decapsulation invokes decryption (which finally encodes the message), directly followed by encryption, which decodes the same message, cf. Algorithm 4. We found in our experiments that the leakage of the decoding function (Algorithm 6) was more suitable for SPA, as the mask takes values with strongly differing HW. Thus, in the following we show how this leakage can be used to recover the message and subsequently the long-term secret-key.

4.2 Chosen-Ciphertext SPA of Decoding Function

Inspired by the method of choosing ciphertext presented in [RRCB20], we found that the message recovery attacks go beyond recovery of the shared key - they can also be used for recovering the secret-key under a chosen-ciphertext attack. Our strategy is as follows:

1. Choose ciphertext to reveal a "strong relationship" between the secret-key and the message (*i.e.*, the relation between five different coefficient choices and the respective message values);
2. recover message from the side-channel leakage during KEM decapsulation; and
3. use the "strong relationship" to recover the secret-key based on few recovered messages.

The authors of [RRCB20] use a similar strategy, but focus on the error correcting codes or FO transformation and recover the secret polynomial one coefficient at a time via the distinguishability of one message bit. Their method requires 7,680 traces. We propose a more efficient method that requires only eight traces to recover the full secret-key in Kyber512 KEM for the `pqm4` ARM-specific implementation at `-O0`. Further, we show that with only moderate increase in the number of traces, this approach can also be applied to the implementation compiled with maximal optimization at `-O3`.

The reduced traces requirement of our approach stems from the fact that we focus on the leakage of simple, low-level arithmetic to recover the message one bit at a time. We then use specifically crafted ciphertexts to recover the long-term secret-key from that. In contrast, [RRCB20] targets the leakage of a non-linear hash or error-correcting

code, resulting in the leakage being influenced by multiple coefficients and thus harder to distinguish with a small number of traces.

4.2.1 Constructing Chosen Ciphertext

Unlike the ciphertext construction method in Section 3, here we choose special $\mathbf{u}$ and v so that each message bit acts as a binary classifier that can distinguish the five possible values for the respective coefficient of the secret-key $\mathbf{s}$. The authors of [RRCB20] used an iterative randomized search algorithm to find ciphertexts meeting their criteria. Here, we refine this principle in order to determine the range ciphertext.

Recall that even if we recover the message m based on the decoding function, we still know the values used in the preceding encoding, because the latter produces the message m. From Algorithm 5 we know that

$$t_{8i+j} = \begin{cases} 1 & \text{if } (q-1)/4 < \text{coeffs}\,[8 \cdot i + j] < (3q+1)/4 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where t_{8i+j} is the $(8i+j)$'th bit of m. Also, note that coeffs [] is the output of:

$$\text{coeffs} = v - \text{NTT}^{-1}\left(\hat{\mathbf{s}}^T \circ \text{NTT}\left(\mathbf{u}\right)\right) = v - s_0 \cdot u_0 - s_1 \cdot u_1 = v - s_0 \cdot u_0 (\text{let } u_1 = 0) \tag{4}$$

From this and because we control u_0 and v, we know that the value of t_{8i+j} (*i.e.*, one bit of the message) entirely depends on the value of one single coefficient of the secret-key. Again, note that as explained in Section 3, we have to choose possible values of u_0 and v out of the respective sets of fixed points U and V.

For example, if we choose $\mathbf{u}^T = (211, 0)$ and $v = \sum_{k=0}^{255} 416 \cdot x^k$, then $t_{8i+j} = 1$ iff $s_0\,[8i+j] = -2$ (because all other possible values for the secret coefficient lead to the result being outside the interval [833, 2496]).

Similarly, we can find u_0, v such that $t_{8i+j} = 1$ iff $s_0\,[8i+j] = -2$ or -1, $s_0\,[8i+j] = 2$, and $s_0\,[8i+j] = 2$ or 1. Going through all possible values of u_0 and v, we find that there are 29 different m-distributions on the 5 possible coefficient values. We regard each distribution (except the cases where the m value is always zero or always one for all possible coefficient values) as a binary classifier which can divide the 5 possible coefficient values into two categories based on one bit value of m. Figure 7 illustrates this for fixed $v = \sum_{k=0}^{255} 416 \cdot x^k$ and different values of u_0.

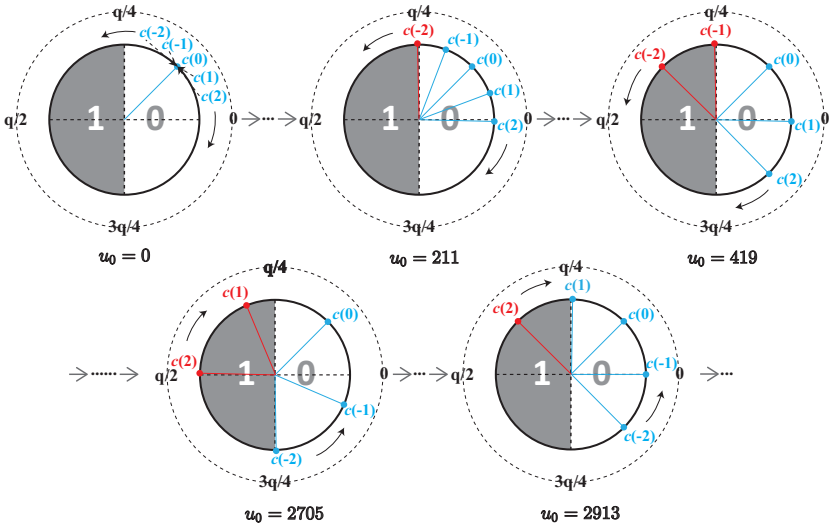

Figure 7: Some examples of changes of m-distribution over the variation of u_0 with a fixed $v = \sum_{k=0}^{255} 416 \cdot x^k$ (We use c as abbreviation for `coeff`)

Similarly, Table 1 in Appendix A gives the resulting classifiers for a fixed $v = \sum_{k=0}^{255} 416 \cdot x^k$ and u_0 being a constant within the indicated intervals (only using values $\in U$). We can utilise this in a One-versus-the-Rest (OvR) classifier, a basic method from multi-class classification [Bis07]. Even though there are not many OvR classifiers in our case, we can build equivalent classifiers. For example, in Table 1, when $u_0 \in [211, 416]$, the distribution is a OvR classifier. If we combine this with $[419, 624]$, we have an equivalent OvR classifier for (0, 1, 0, 0, 0). Similarly, we can construct (0, 0, 0, 0, 1) and (0, 0, 0, 1, 0). With these four different distributions (the final four cases in Figure 7, *i.e.*, $u_0 = 211, 419, 2705, 2913$), we can recover all the five different coefficients.

Note that other combinations of the 27 non-trivial distributions can also achieve the goal of distinguishing five possible values of coefficients. For brevity, in the following we only use the above four ciphertexts as chosen-ciphertexts without losing universality.

If we can recover the message (in other words: distinguish $t = 1$ from $t = 0$), we can thus recover half of the secret-key (s_0) with only 4 different chosen ciphertexts. More precisely, let $\mathbf{m}^{u_0}$ denote the vector of recovered message bits for a given ciphertext u_0 (and $v = \sum_{k=0}^{255} 416 \cdot x^k$). Then we have the following relation between k-th bit $\mathbf{m}_k^{u_0}$ and the k-th coefficient $s_0[k]$ of the secret polynomial s_0:

$$\begin{cases} \mathbf{m}_k^{211} = 1 & \text{iff } s_0[k] = -2 \\ \mathbf{m}_k^{419} - \mathbf{m}_k^{211} = 1 & \text{iff } s_0[k] = -1 \\ \mathbf{m}_k^{2705} - \mathbf{m}_k^{2913} = 1 & \text{iff } s_0[k] = 1 \\ \mathbf{m}_k^{2913} = 1 & \text{iff } s_0[k] = 2 \\ \text{else} & \text{iff } s_0[k] = 0 \end{cases}$$

Hence, we get the coefficient vector $\mathbf{s_0}$ of the polynomial s_0 as

$$\mathbf{s_0} = (-2) \cdot \mathbf{m}^{211} + (-1) \cdot (\mathbf{m}^{419} - \mathbf{m}^{211}) + 1 \cdot (\mathbf{m}^{2705} - \mathbf{m}^{2913}) + 2 \cdot \mathbf{m}^{2913}$$

The second half s_1 can be recovered with four more traces, using the same values but for u_1, setting $u_0 = 0$. Next, we practically demonstrate how the message can be recovered for different optimization levels (`-O0` and `-O3`).

4.2.2 Message Recovery for `-O0`

From Algorithm 6, it is clear that the value of mask and the output polynomial coefficients differ depending on the respective message bit (mask = 0xffff and $r\,[8 \cdot i + j] = (q+1)/2$ iff $((\text{msg}[i] \gg j)\ \&\ 1) = 1$). As evident in Figure 8, the difference between a message bit being 0 or 1 is immediately visible in the respective trace when `poly_frommsg()` is compiled at `-O0`. Our algorithm for automated message recovery proceeds in two steps: First, we determine the set of PoI using local extreme value search based on two reference traces (where all message bits are 0 and 1, respectively), as shown in Algorithm 7. This step also computes a threshold T to distinguish message bit values.

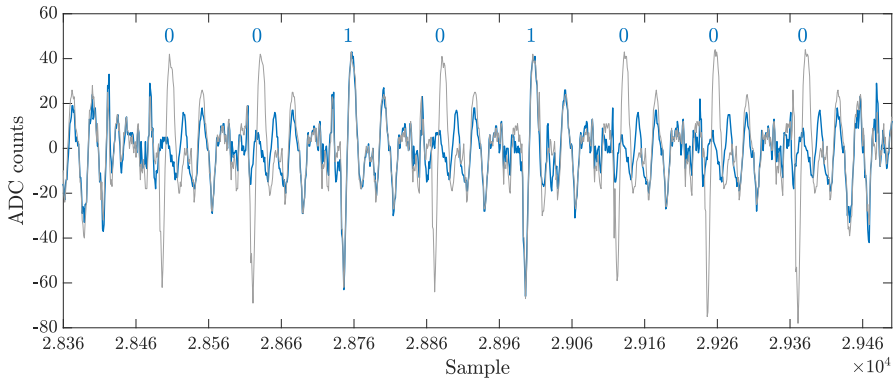

Figure 8: Example trace at `-O0` (blue) showing the differences between message bit = 0 and 1. Reference trace r_1 (with all bits set) in gray.

Algorithm 7 PoI detection and threshold computation for compiler optimization `-O0`.

Input: Reference trace where all message bits are 0:$r_0(t)$, reference trace where all message bits are 1:$r_1(t)$

Find PoI: Perform local extrema search on r_1 to find 256 local maxima and minima:
$PoI_{max} = \{PoI_{max}(0), PoI_{max}(1), \ldots, PoI_{max}(255)\}$
$PoI_{min} = \{PoI_{min}(0), PoI_{min}(1), \ldots, PoI_{min}(255)\}$
Find average difference between maxima and minima:
for $i = 0 \ldots 255$ **do**
  $a_0(i) = r_0(PoI_{max}(i)) - r_0(PoI_{min}(i))$
  $a_1(i) = r_1(PoI_{max}(i)) - r_1(PoI_{min}(i))$
end for
Compute threshold $T = 0.5 \cdot \left(\sum_{j=0}^{255} a_0(i)/256 + \sum_{j=0}^{255} a_1(i)/256\right)$
return PoI_{min}, PoI_{max}, T

In the actual attack phase, given a target trace $p(t)$, Algorithm 8 reconstructs the message by comparing the difference at the previously determined PoI to the threshold T. We found that our message recovery process can correctly recover 100% of the message bits from a single trace. Using the methods from Section 4.2.1, we can hence recover the full secret-key with eight traces in total.

Algorithm 8 Message recovery for compiler optimization `-O0`.

Input: PoI PoI_{min}, PoI_{max}, threshold T, trace to be analyzed $p(t)$

msg = (0, 0, ..., 0)
for $i = 0 \ldots 255$ **do**
  $\delta(i) = p(PoI_{max}(i)) - p(PoI_{min}(i))$
  if $\delta(i) > T$ **then**
    msg$[i] \leftarrow 1$
  else
    msg$[i] \leftarrow 0$
  end if
end for
return msg

Note that our algorithm requires a one-time profiling step (Algorithm 7) , typically

carried out using a profiling device. As the PoI are points in time, they are unlikely to differ between different profiling and target device. On the other hand, the threshold T is an amplitude quantity and hence might vary depending on device, measurement setup and so on. However, Algorithm 8 can be easily modified to compute the threshold in real-time using only the target trace. This can be achieved by clustering the observed δ at the PoI into two groups, and finding the best threshold to distinguish these clusters (similar to the approach in Section 3).

Alternatively, the profiling can also be carried out with the actual target device, avoiding any potential issues due to portability of the PoI and T. For this, the adversary crafts special ciphertexts as inputs to `poly_frommsg()` in the initial decoding step (Line 2 in Algorithm 2) within the decryption in KEM decapsulation. For instance, setting $v = \sum_{k=0}^{255} 1665 \cdot x^k$ and $u_0 = u_1 = 0$ leads to an all-one message in this step (details in Table 2 in Appendix A). Hence, our algorithm is robust towards portability issues that typically affect template attacks, which heavily rely on precise amplitude profiling.

4.2.3 Message Recovery for `-O3`

When compiled at `-O3`, the leakage of the coefficient-wise processing of the message is less pronounced compared to `-O0`. To be able to employ a similar approach as in Section 4.2.2, we rely on averaging of N traces for the same chosen ciphertext (*i.e.*, $\mathbf{u}$ and v). Again, we use an initial profiling step to determine PoI and threshold. As explained in Section 4.2.2, this can be done using a dedicated profiling device or on the target device through appropriately chosen ciphertext. In contrast to `-O0`, we use mean reference traces averaged over $R = 400$ traces with all-zero/all-one message each. Precisely, the profiling is shown in Algorithm 9.

Algorithm 9 PoI detection and threshold computation for `-O3`.

Input: R reference traces where all message bits are 0: $r_{0,i}(t)$, R reference traces where all message bits are 1: $r_{1,i}(t)$

1: $\overline{r}_0(t) = \left(\sum_{i=1}^{R} r_{0,i}(t)\right)/R$
2: $\overline{r}_1(t) = \left(\sum_{i=1}^{R} r_{1,i}(t)\right)/R$
3: $\Delta_r(t) = \overline{r}_1(t) - \overline{r}_0(t)$
4: Find PoI: Perform downward peak detection on $\Delta_r(t)$ to find 256 local minima:
5: $PoI_{min} = \{PoI_{min}(0), PoI_{min}(1), \ldots, PoI_{min}(255)\}$
6: Compute threshold $T = 0.5 \cdot \left(\sum_{j=0}^{255} \Delta_r(PoI(j))\right)/256$
7: **return** PoI_{min}, T

The actual message recovery continues in a similar way to Algorithm 8, however, it operates on an average over N traces $p_i(t)$ as follows. We first compute the average trace as: $\overline{p}(t) = \left(\sum_{i=1}^{N} p_i(t)\right)/N$. Then, rather than focusing on the difference between minimum and maximum (cf. Line 3 in Algorithm 8), we compute the $\delta(i)$ at each PoI as the difference between $\overline{p}(t)$ and the average reference trace for the "all-zero" case $\overline{r}_0(t)$, *i.e.*, $\delta(i) = \overline{p}(PoI(i)) - \overline{r}_0(PoI(i))$. Also, because T is negative, a 1 is recovered if $\delta(i) < T$, and a 0 otherwise.

The success rate (*i.e.*, the percentage of correctly recovered secret coefficients based on the message bits) of our method over the total number of traces, *i.e.*, $8 \cdot N$, is shown in Figure 10. In all cases, the reference traces were computed for $R = 400$, and the experiment was carried out for 16 randomly chosen secret-keys. As evident in Figure 10, the success rate stabilizes at 100% after approximately 960 traces in total (*i.e.*, 120 averages per trace). Furthermore, there is also a trade-off with partial exhaustive search, e.g., the success rate

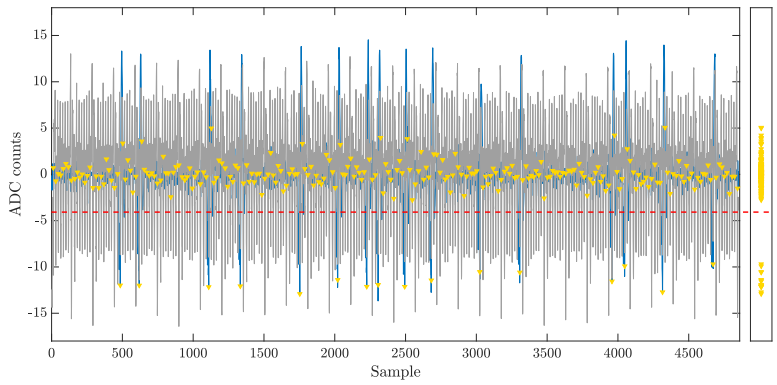

Figure 9: Example average difference trace δ at `-O3` (blue) showing the differences between 0 and 1 bits. Difference-of-means reference trace Δ_r in gray.

reaches 98% (*i.e.*, recovering 502 of 512 coefficients) after 184 traces (*i.e.*, 23 averages per trace). This translates to a remaining brute-force effort of 5^{10}, *i.e.*, approximately $2^{23.22}$. Compared to the previous research of [RRCB20], which requires 7,680 traces for full-key recovery for Kyber512, our method succeeds with substantially less traces. In contrast to [ACLZ20], our approach avoids the need for templates in the message recovery phase.

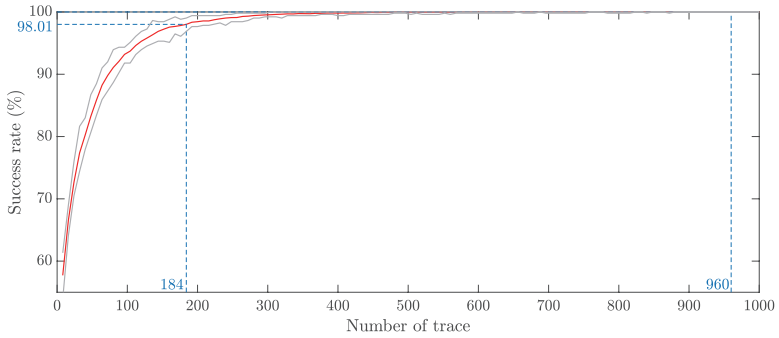

Figure 10: Average (red) and min/max (gray) percentage of correctly recovered secret-key coefficients (success rate) vs. total number of traces. Average computed over 16 random secret-keys. Success rate reaches 98% after 184 traces and stable at 100% after 960 traces.

5 Conclusion and Countermeasures

Our work presented an efficient EM side-channel-assisted CCA against IND-CCA Kyber KEM. We identified several fundamental building blocks in Kyber that leak sensitive information via side-channels. Thereafter, we showed how a side-channel attacker could construct malicious ciphertexts to amplify such leakage and then extract the long-term secret-key using a very small number of traces.

Summary We experimentally validated our attack strategies on both reference and optimized implementations of Kyber (specifically Kyber512) from the popular `pqm4` library. For the reference implementation, a direct key recovery is possible using only four traces with 100% success rate. For the assembly-optimized implementation of Kyber, we observed that using a small number of traces we could not perform key recovery *directly* but could perform

message recovery with high accuracy. Finally, we showed that the recovered message has a strong relationship with the long-term key and by exploiting this relationship, an attacker could easily compute the long-term secret-key. Our experimental results show that our attack needs only 8 and 960 traces at `-O0` and `-O3` compiler-optimization levels respectively to recover the long-term secret-key from the assembly-optimized implementation of Kyber.

Extension to other PQC Schemes Because the fundamental building blocks are similar in all lattice-based public-key encryption or encapsulation schemes (although their algorithms and implementations may vary), it might be possible to extend our attack techniques to the other PQC candidate schemes with some adaptations. The complexity level of an extended attack may vary scheme to scheme—for example, in NewHope, our message-recovery attacks to extract the long-term secret-key (Section 4.2.1) cannot be applied straightforwardly, as one bit of the message is influenced by two secret coefficients. Hence, constructing similar efficient CCA side-channel techniques is an interesting aspect for future work.

Countermeasures Our EM-based CCA accumulates side-channel leakage from computations that involve a *long-term* secret-key. Hence, mounting such an attack becomes impractical on applications where the secret-key is not reused. When the secret-key needs to be reused many times, masking-based countermeasures can be applied to protect the secret-key. Such countermeasures work by splitting the secret in random shares and thereafter randomizing the entire decryption or decapsulation. Several generic masking countermeasures [RRVV15, RdCR+16, RRdC+16, OSPG18] for lattice-based public-key encryption have been proposed against differential power attacks. However, the application of masking typically increases the execution time of decryption or decapsulation by some factor. Recently Beirendonck et al. [BDK+20] showed that it is possible to reduce this overhead if the masking countermeasure is customized targeting a specific public-key scheme. They showed that an optimized, masked decapsulation becomes only 2.5 times slower with respect to unmasked decapsulation for Saber [DKRV19]. We believe that further study in this direction will produce more optimized masking schemes in the future.

A less computationally-expensive countermeasure could be to detect and then discard malicious ciphertexts before starting any computation involving the long-term secret-key. In our attack, the coefficients of the chosen-ciphertext polynomials are fabricated to satisfy a specific structure such that the EM leakage reveals information about the secret. Genuine ciphertexts are essentially LWE samples, and hence they are indistinguishable from uniformly random samples. By measuring the entropy of the received ciphertext, we could detect and then discard specially structured (*i.e.*, low-entropy) malicious ciphertexts before starting a *real* decapsulation. However, such a countermeasure needs further analysis and would additionally cause false-positive cases.

6 Acknowledgements

The research is partially funded by the Engineering and Physical Sciences Research Council (EPSRC) under grant EP/R012598/1 and by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM). The work was done during the visit of the first author to the University of Birmingham. The visit was funded by the China Scholarship Council (CSC) under Grant No. 201906020096.

References

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Jimmy Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Michael Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew Harrigan, Michael Hartmann, Alan Ho, Markus Rudolf Hoffmann, Trent Huang, Travis Humble, Sergei Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, Dave Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod Ryan McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin Jeffery Sung, Matt Trevithick, Amit Vainsencher, Benjamin Villalonga, Ted White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John Martinis. Quantum Supremacy using a Programmable Superconducting Processor. *Nature*, 574:505–510, 2019.

[ACLZ20] Dorian Amiet, Andreas Curiger, Lukas Leuenberger, and Paul Zbinden. Defeating NewHope with a Single Trace. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2020.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum Key Exchange - A New Hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

[ATT+18] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal Side-channel Vulnerabilities of Post-quantum Key Exchange Protocols. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018*, pages 81–88. IEEE Computer Society, 2018.

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: Reducing Attack Surface at Low Cost. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, 2017.

[BDHD+19] Ciprian Băetu, F Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse Attacks on Post-quantum Cryptosystems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 747–776. Springer, 2019.

[BDK+18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 353–367, 2018.

[BDK+20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A Side-Channel Resistant Implementation of SABER. *IACR Cryptol. ePrint Arch.*, 2020:733, 2020.

[BGRR19] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. Assessment of the key-reuse resilience of NewHope. In *Cryptographers' Track at the RSA Conference*, pages 272–292. Springer, 2019.

[Bis07] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition.* Information science and statistics. Springer, 2007.

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje-eddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, volume 11627 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2019.

[DCQ19] Jintai Ding, Chi Cheng, and Yue Qin. A Simple Key Reuse Attack on LWE and Ring LWE Encryption Schemes as Key Encapsulation Mechanisms (KEMs). *IACR Cryptology ePrint Archive*, 2019:271, 2019.

[DKRV19] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Proposal to NIST PQC Standardization, Round2, 2019. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/round-2-submissions.

[DKSRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. *Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM*, volume 10831, page 282–305. Springer International Publishing, 2018.

[DLL+18] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, Feb. 2018.

[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, London, UK, November 11, 2019*, pages 2–9. ACM, 2019.

[EG12] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *Journal of Cryptographic Engineering*, 2:63–74, 2012.

[Flu16] Scott R Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 537–554, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[HCY20] Wei-Lun Huang, Jiun-Peng Chen, and Bo-Yin Yang. Power analysis on NTRU prime. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):123–151, 2020.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[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.

[LLZ+18] Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, and Bao Li. LAC: practical ring-lwe based public-key encryption with byte-level modulus. *IACR Cryptol. ePrint Arch.*, 2018:1009, 2018.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[Nat20] National Institute of Standards and Technology. Post-quantum cryptography, Created January 03, 2017, Updated June 24, 2020. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-Secure and Masked Ring-LWE Implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[QCD19] Yue Qin, Chi Cheng, and Jintai Ding. A Complete and Optimized Key Mismatch Attack on NIST Candidate NewHope. *IACR ePrint Archive*, page 435, 2019.

[RBRC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. Drop by Drop you break the rock - Exploiting generic vulnerabilities in Lattice-based PKE/KEMs using EM-based Physical Attacks. *IACR Cryptol. ePrint Arch.*, 2020:549, 2020.

[RdCR+16] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-LWE masking. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 233–244, 2016.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020.

[RRdC+16] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-lwe. *J. Cryptogr. Eng.*, 6(2):139–153, 2016.

[RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A Masked Ring-LWE Implementation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 683–702. Springer, 2015.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SAB+19] P Schwabe, R Avanzi, J Bos, L Ducas, E Kiltz, T Lepoint, V Lyubashevsky, JM Schanck, G Seiler, and D Stehle. CRYSTALS-Kyber–Algorithm Specifications And Supporting Documentation. *NIST Technical Report*, 2019.

[Sho97] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997.

[STM16] STMicroelectronics. *STM32F405xx STM32F407xx Datasheet*, 9 2016. Rev 8.

A Message Distributions for Different Chosen Ciphertexts

Table 1: m-distributions for different intervals of $u_0\,[0]$ with fixed $v = \sum_{k=0}^{255} 416 \cdot x^k$.

t coeff. of s / u_0	-2	-1	0	1	2
$[0, 208]$	0	0	0	0	0
$[211, 416]$	1	0	0	0	0
$[419, 624]$	1	1	0	0	0
$[627, 1040]$	1	1	0	0	1
$[1044, 1248]$	0	1	0	0	1
$[1252, 1456]$	0	1	0	1	1
$[1460, 1869]$	0	1	0	1	0
$[1873, 2077]$	1	1	0	1	0
$[2081, 2285]$	1	0	0	1	0
$[2289, 2702]$	1	0	0	1	1
$[2705, 2910]$	0	0	0	1	1
$[2913, 3118]$	0	0	0	0	1
$[3121, 3326]$	0	0	0	0	0

Table 2: m-distributions for different intervals of $u_0\,[0]$ with fixed $v = \sum_{k=0}^{255} 1665 \cdot x^k$.

t coeff. of s / u_0	-2	-1	0	1	2
$[0, 413]$	1	1	1	1	1
416	0	1	1	1	1
$[419, 829]$	0	1	1	1	0
832	0	0	1	1	0
$[836, 1248]$	0	0	1	0	0
$[1252, 2077]$	1	0	1	0	1
$[2081, 2493]$	0	0	1	0	0
2497	0	1	1	0	0
$[2500, 2910]$	0	1	1	1	0
2913	1	1	1	1	0
$[2916, 3326]$	1	1	1	1	1