IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 795–818. DOI:10.46586/tches.v2024.i2.795-818

Defeating Low-Cost Countermeasures against Side-Channel Attacks in Lattice-based Encryption

A Case Study on Crystals-Kyber

Prasanna Ravi[1], Thales Paiva[2,3], Dirmanto Jap[1], Jan-Pieter D'Anvers[4] and Shivam Bhasin[1]

[1] Temasek Laboratories, Nanyang Technological University, Singapore, Singapore
[2] Fundep, Belo Horizonte, Brazil
[3] CASNAV, Rio de Janeiro, Brazil
[4] imec-COSIC KU Leuven, Kasteelpark Arenberg 10 - bus 2452, 3001 Leuven, Belgium
prasanna.ravi@ntu.edu.sg thalespaiva@gmail.com djap@ntu.edu.sg
janpieter.danvers@esat.kuleuven.be sbhasin@ntu.edu.sg

Abstract. In an effort to circumvent the high cost of standard countermeasures against side-channel attacks in post-quantum cryptography, some works have developed low-cost detection-based countermeasures. These countermeasures try to detect maliciously generated input ciphertexts and react to them by discarding the ciphertext or secret key. In this work, we take a look at two previously proposed low-cost countermeasures: the ciphertext sanity check and the decapsulation failure check, and demonstrate successful attacks on these schemes. We show that the first countermeasure can be broken with little to no overhead, while the second countermeasure requires a more elaborate attack strategy that relies on valid chosen ciphertexts. Thus, in this work, we propose the first chosen-ciphertext based side-channel attack that only relies on valid ciphertexts for key recovery. As part of this attack, a third contribution of our paper is an improved solver that retrieves the secret key from linear inequalities constructed using side-channel leakage from the decryption procedure. Our solver is an improvement over the state-of-the-art Belief Propagation solvers by Pessl and Prokop, and later Delvaux. Our method is simpler, easier to understand and has lower computational complexity, while needing less than half the inequalities compared to previous methods.

Keywords: Lattice-based cryptography · Side-Channel Attack · Kyber · Key Encapsulation Mechanism · Chosen Ciphertext Attack

1 Introduction

In July 2022, a new Key Encapsulation Mechanism (KEM) and 3 digital signature schemes were selected as the first standardized PQC algorithms [AAC+22], a result of an extensive NIST Post-Quantum standardization process. Kyber KEM [ABD+20], based on the well-known Module Learning With Error (MLWE) problem, was the only algorithm standardized for KEMs, and we will soon witness wide-scale adoption of Kyber on a variety of computing devices, including resource-constrained platforms such as embedded microcontrollers [KRSS19,AHKS22]. This naturally makes them susceptible to side-channel attacks, which was also an important consideration during the NIST PQC standardization process [RCDB22]. Thus, several works were reported on SCA of Kyber KEM and proposals for concrete countermeasures [BGR+21,RRD+23,RRCB20].

Received: 2023-10-15 Accepted: 2023-12-15 Published: 2024-03-12

Looking at these attacks, the decapsulation procedure of Kyber KEM typically serves as the main target, and they can all be categorized into two broad categories: (1) Known Ciphertext (KC) based SCA and (2) Chosen Ciphertext based SCA. In KC-based SCA, the attacker only requires the knowledge of inputs to the decapsulation procedure [PPM17, PP19, MWK+22]. However, for CC-based SCA, the attacker must control the input to the decapsulation procedure [RRCB20, XPRO20, BDH+21]. CC-based attacks form the largest category of attacks and are considered more powerful than KC attacks. The reason is that CC-based attacks can exploit leakage from the entire decapsulation procedure for key recovery, while KC-based attacks have a much narrower attack surface for key recovery. This means that protection against CC-based attacks is costlier than KC-based attacks and requires combining masking and shuffling countermeasures [BGR+21, RPBC20]. Thus, CC-based side-channel attacks form the main focus of our work.

CC-based side-channel attacks utilize maliciously crafted ciphertexts for key recovery. Standard countermeasures against these attacks include masking and shuffling, which come at a certain cost. In a different approach, some works have focused on low-cost detection-based countermeasures [XPRO20, RCDB22]. The idea of detection-based countermeasures is to look for maliciously formed ciphertexts. Upon detection of a malicious ciphertext, the ciphertext can be discarded, or the secret key can be refreshed, preventing further exposure towards the attacker. While these countermeasures look attractive for a designer owing to their low cost, we are unaware of a dedicated study of the concrete protection they offer against CC-based side-channel attacks. In this work, we attempt to bridge this gap, we perform a concrete analysis of two such detection countermeasures against CC-based attacks, the (1) Ciphertext Sanity Check and the (2) Decapsulation Failure Check, and demonstrate new types of CC-based attacks are capable of breaking both these countermeasures.

Our Contribution

In this work, we present the following new contributions:

1. We give the first analysis of the effectiveness of low-cost detection-based countermeasures against CC-based side-channel attacks. In particular, we analyze two countermeasures - (1) Ciphertext Sanity Check and (2) Decapsulation Failure Check. We demonstrate novel CC-based side-channel attacks that can break both countermeasures efficiently and apply these attacks in the case of Kyber KEM.

   (a) Our first attack targets the ciphertext sanity check countermeasure, which detects low-entropy ciphertexts used for a large subset of CC-based side-channel attacks. We introduced a novel method to mask malicious low-entropy ciphertexts with the public key to increase their entropy and circumvent the ciphertext sanity check countermeasure. One main advantage of our approach is that, after masking, the attack strategy is the same as the attack strategy without the countermeasure. Our attack works with probability 57.8% and can be restarted if unsuccessful.

   (b) Our second attack targets the decapsulation failure check countermeasure, which refreshes the secret key upon observing a single decapsulation failure. This countermeasure offers concrete protection as all CC-based attacks rely on multiple invalid/malicious ciphertexts for key recovery. In this work, we propose the first CC-based side-channel attack that only relies on valid ciphertexts to counter the decapsulation failure check countermeasure.

2. We develop a novel solver to obtain the secret key from linear inequalities obtained from leakage. This solver improves upon the Belief Propagation approach, which

has been the de-facto approach in prior works [PP21, HPP21, Del22]. Our solver is easier to understand, more efficient to run and can recover the secret key with less than 2× number of linear inequalities compared to the state-of-the-art.

We perform experimental validation of our CC attack using valid ciphertexts, on both the unprotected and masked open-source implementations of Kyber KEM taken from the pqm4 library [KRSS] and mkm4 library [HKL+22] respectively. On the unprotected implementations, we can recover the secret key in ≈ 325 traces for the reference implementation and ≈ 7800 traces for the optimized implementation. We also show that our attack can be adapted to both the shuffled and masked implementations in a straightforward manner. While both shuffling and masking increase the attacker's complexity for key recovery, they do not concretely prevent the attack.

Our work therefore shows that low-cost detection countermeasures can be rendered completely ineffective, and do not offer standalone protection against CC-based side-channel attacks. While these countermeasures are attractive for designers, our work encourages more study towards the development and analysis of new detection-based countermeasures against CC-based side-channel attacks.

Availability of Software

All the software used in this work is placed in the public domain in the following link: https://github.com/thalespaiva/sca_on_kyber_lcc.

2 Preliminaries

2.1 Notations

For any prime q, we let $\mathbb{Z}_q$ denote the field of integers modulo q. Let R_q be the polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$, then R_q^k is a module of dimension k. Polynomials $a \in R_q$ are denoted using lower case letters. Vectors $\mathbf{a} \in R_q^k$ and matrices $\mathbf{A} \in R_q^{k\times k}$ are denoted in bold using lower and upper case, respectively. When $\mathbf{u}, \mathbf{v} \in R_q^k$, we let $\langle \mathbf{u}, \mathbf{v} \rangle \in R_q$ denote their dot product. The ith coefficient of a polynomial a is denoted as $a[i]$. For any set V, we write $v \xleftarrow{\$} \mathcal{U}(V)$ to denote that v is chosen uniformly at random from V. We denote as χ the Centered Binomial Distribution (CBD) with range $[-\eta, \eta]$. In this case, we abuse notation and let $\mathbf{a} \xleftarrow{\$} \chi(R_q^k)$ mean that each coefficient of each polynomial of $\mathbf{a} \in R_q^k$ is drawn according to χ. Rounding a coefficient of a polynomial $a \in R_q$ from modulus q to modulus p is denoted as $\lfloor a \rceil_{p\to q}$. Furthermore, rounding a floating point value $a \in \mathbb{R}$ to the nearest integer is denoted as $\lfloor a \rceil$, with ties being rounded up.

Let `poly_to_vec` be the function that, given a polynomial $a \in R_q$, returns the vector in $\mathbb{Z}_q^n$ consisting of the n coefficients of a. Furthermore, let $\texttt{negashift}_i$ be the function that returns a negacyclic shift of a vector by i positions. That is, if $a = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$, then $\texttt{negashift}_i(a) = \left[a_i, \ldots, a_0, -a_{n-1}, \ldots, -a_{i+1}\right]$. Using this notation, we can express the coefficients of the product of polynomials a and b in the negacyclic ring R_q in vector form as

$$\texttt{poly\_to\_vec}(ab)[i] = \langle \texttt{negashift}_i(a), \texttt{poly\_to\_vec}(b) \rangle .$$

If we extend these two notations for vectors of polynomials $\mathbf{u}, \mathbf{v} \in R_q^k$ by applying the `poly_to_vec` and `negashift` to each of the k polynomials, we get the analogous property

$$\texttt{poly\_to\_vec}(\langle \mathbf{u}, \mathbf{v} \rangle)[i] = \langle \texttt{negashift}_i(\mathbf{u}), \texttt{poly\_to\_vec}(\mathbf{v}) \rangle . \tag{1}$$

Algorithm 1: CPAPKE.KeyGen.

$\mathbf{A} \xleftarrow{\$} \mathcal{U}(R_q^{k \times k})$
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \chi(R_q^k) \times \chi(R_q^k)$
$\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
return $pk := (\mathbf{A}, \mathbf{b}), sk := \mathbf{s}$

Algorithm 2: CPAPKE.Enc.

Input: $pk = (\mathbf{A}, \mathbf{b})$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
$(\mathbf{r}, \mathbf{e}_1, e_2) \xleftarrow{r} \chi(R_q^k) \times \chi(R_q^k) \times \chi(R_q)$
$\mathbf{u} \leftarrow \mathbf{A}^\top \cdot \mathbf{r} + \mathbf{e}_1$
$v \leftarrow \langle \mathbf{b}, \mathbf{r} \rangle + e_2 + \left\lfloor \frac{q}{2} \right\rceil \cdot m$
$\mathbf{c}_1 \leftarrow \lfloor \mathbf{u} \rceil_{q \to p}$
$c_2 \leftarrow \lfloor v \rceil_{q \to T}$
return $c := (\mathbf{c}_1, c_2)$

Algorithm 3: CPAPKE.Dec.

Input: $sk = s$
Input: $c = (\mathbf{c_1}, \mathbf{c_2})$
$\mathbf{u}' \leftarrow \lfloor \mathbf{c}_1 \rceil_{p \to q}$
$v' \leftarrow \lfloor c_2 \rceil_{T \to q}$
$m' \leftarrow v' - \langle \mathbf{s}, \mathbf{u}' \rangle$
$m \leftarrow \lfloor m' \rceil_{q \to 2}$
return m

2.2 Kyber KEM

Kyber is a CCA-secure KEM based on the hardness of the Module-Learning With Errors problem (MLWE) [ABD+20]. It offers parameter sets for three NIST security levels 1, 3, and 5, named Kyber512, Kyber768, and Kyber1024 respectively. All instantiations operate over the same anti-cyclic polynomial ring $R_q = \mathbb{Z}_q/(X^n+1)$ with a prime modulus $q = 3329$ and degree $n = 256$. This ring is used to build the module R_q^k, where $k = 2, 3$ or 4, for security levels 1, 3, and 5. The CCA-secure Kyber contains, at its core, an IND-CPA secure PKE scheme, which is reviewed next.

2.2.1 IND-CPA Secure Kyber PKE

The IND-CPA secure Kyber PKE scheme consists of three procedures: CPAPKE.KeyGen, CPAPKE.Encrypt and CPAPKE.Decrypt. Algorithms 1, 2 and 3 show simplified descriptions of the corresponding procedures.

Key generation: The key generation procedure shown in Algorithm 1 essentially involves the creation of an LWE instance. The coefficients of the secret key $\mathbf{s} \in R_q^k$ and error $\mathbf{e} \in R_q^k$ are sampled from the narrow centered binomial distribution χ, while coefficients of $\mathbf{A}$ are sampled from the uniform distribution $\mathcal{U}$. The MLWE instance is computed as $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$. The public key is the pair $(\mathbf{A}, \mathbf{b})$ while the secret key is $\mathbf{s}$.

Encryption: The encryption procedure shown in Algorithm 2 first samples $\mathbf{r} \in (R_q^k)$, $\mathbf{e}_1 \in R_q^k$ and $e_2 \in R_q$ from χ, which, together with the public key $pk = (\mathbf{A}, \mathbf{b})$, are used to compute two LWE components: $\mathbf{u} \in R_q^k$ and $v \in R_q$. Polynomial v is computed using the 256 bit message $m \in \{0,1\}^{256}$, for which it is encoded to a corresponding polynomial in R_q by simply multiplying each bit of the message by $\lfloor q/2 \rceil$. The vector $(\mathbf{u}, v)$ undergo coefficient-wise compression after which they serve as the ciphertext outputs $(\mathbf{c}_1, c_2) \in R_q^k \times R_q$.

Decryption: The decryption procedure shown in Algorithm 3 (CPAPKE.Dec) involves the computation of the noisy message polynomial $m' = (v' - \mathbf{s}^T \cdot \mathbf{u}') \in R_q$, where $\mathbf{u}' \in R_q^k$ and $v' \in R_q$ are decompressed versions of values $(\mathbf{u}, v)$ computed during encryption.

The compression and decompression introduces a certain error in $\mathbf{u}$ and v, which we denote as $\Delta\mathbf{u} = \mathbf{u}' - \mathbf{u}$ and $\Delta v = v' - v$, respectively. The noisy message polynomial m' is then decoded to the correct message $m \in \{0,1\}^{256}$ by simply decoding each of its coefficients $m'[i]$ to 0 or to 1, depending if $m'[i]$ is closer to 0 or to $q/2$.

Algorithm 4: CCAKEM.KeyGen.

$z \xleftarrow{\$} \{0,1\}^{256}$
$(pk, sk') = \text{CPA.KeyGen}()$
$sk = (sk'||pk||\text{H}(pk)||z)$
return pk, sk

Algorithm 5: CCAKEM.Encaps.

Input: Public key of CCAKEM pk
$m \xleftarrow{\$} \{0,1\}^{256}$
$m \leftarrow \text{H}(m)$
$(\bar{K}, r) = \text{G}(m||\text{H}(pk))$
$c = \text{CPA.Enc}(pk, m, r)$
$K = \text{KDF}(\bar{K}||\text{H}(c))$
return c, K

Algorithm 6: CCAKEM.Decaps.

Input: Ciphertext of CCAKEM c
Input: Secret key of CCAKEM sk
Extract $(sk'||pk||\text{H}(pk)||z)$ from sk
$m' = \text{CPA.Dec}(sk', c)$
$(\bar{K}', r') = \text{G}(m'||\text{H}(pk))$
$c' = \text{CPA.Enc}(pk, m', r')$
if $c = c'$ **then**
  $K = \text{KDF}(\bar{K}'||\text{H}(c))$
else
  $K = \text{KDF}(z||\text{H}(c))$
return K

Let us review why the decryption works. Notice that the noisy message polynomial m' can be represented as follows

$$m' = v' - \langle \mathbf{s}, \mathbf{u}' \rangle = (v + \Delta v) - \langle \mathbf{s}, \mathbf{u} + \Delta \mathbf{u} \rangle .$$

After expanding each term and simplifying it, we get that $m' = m \left\lfloor \frac{q}{2} \right\rceil + \Delta m$, where

$$\Delta m = \langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 + \Delta \mathbf{u} \rangle + e_2 + \Delta v. \tag{2}$$

Notice that Δm is made of components sampled from the narrow distribution χ, and therefore m' can be seen as a noisy version of the original message polynomial $\left\lfloor \frac{q}{2} \right\rceil \cdot m$. Kyber security parameters are responsible for ensuring that the coefficients in Δm are small enough so that decryption errors occur only with negligible probability.

2.2.2 IND-CCA Secure Kyber KEM

The IND-CPA secure Kyber KEM can be transformed into IND-CCA secure KEM using the well-known Fujisaki Okamoto (FO) transform [FO99]. This involves instantiation of the CPAPKE.Enc, CPAPKE.Dec procedures of Kyber PKE scheme, along with several instances of hash functions resulting in IND-CCA secure encapsulation (CCAKEM.Encaps) and decapsulation (CCAKEM.Decaps) procedures respectively. Algorithm 4 - 6 supplies the details. The main idea is that the randomness required for CPAPKE.Enc is made explicit through a seed r derived from the message m and public key pk. This enables the decapsulation procedure to retrieve the message m from ciphertext ct, compute the seed r, and re-compute the ciphertext ct'.

Subsequently, the recomputed ciphertext ct' is compared with the received ciphertext ct, and this comparison only succeeds for valid ciphertexts and fails for invalid ciphertexts with a very high probability. In theory, the FO transform helps check the validity of ciphertexts through a re-encryption procedure after decryption. Thus, with a very high probability, the attacker only sees decapsulation failures for invalid ciphertexts. This provides strong *theoretical* security guarantees against chosen-ciphertext attacks.

2.3 Prior Works

The decapsulation procedure of Kyber KEM has been subjected to a wide variety of SCA for key recovery [BDH+21, RRCB20, HHP+21, RRD+23]. These attacks can be broadly classified into two categories, based on the knowledge/control over the ciphertexts manipulated by the decapsulation procedure: (1) Known Ciphertext (KC) based SCA and (2) Chosen Ciphertext (CC) based SCA.

Known Ciphertext (KC) based SCA: The dot product between the ciphertext component $\mathbf{u} \in R_q^k$ and the secret $\mathbf{s} \in R_q^k$ (Line 3 in the CPAPKE.Dec procedure in Alg. 6) has been the sole target for KC-based SCA, as it directly manipulates the secret key. There are single trace template style attacks targeting the INTT operation [PPM17, PP19] as well as the pointwise multiplication operation [YRZ+23, BBB+23]. There are also multiple trace attacks [MWK+22] based on the well-known Correlation Power Analysis (CPA), which require ≈ 200 traces for key recovery. Since the polynomial multiplication operation has been heavily targeted by side-channel attacks, the designer might incorporate additional countermeasures to mitigate potential key recovery.

Chosen Ciphertext (CC) based SCA: The modus operandi of side-channel attacks using chosen ciphertexts [BDH+21, RRCB20, RRD+23, UXT+22] is given as follows. The attacker crafts specially structured malicious ciphertexts such that their corresponding decrypted message m (unknown to the attacker) becomes related to the secret key sk (or a part of it). This ensures that leakage from the entire FO transform can be utilized as an *oracle* for key recovery. These attacks can be broadly classified into three types: (1) Plaintext Checking (PC) Oracle-based SCA [RRCB20, UXT+22, TUX+23], (2) Full-Decryption (FD) Oracle-Based SCA [XPRO20], and (3) Decryption-Failure (DF) Oracle-Based SCA [BDH+21].

2.3.1 Comparing KC-based CCA and CC-based SCA

We observe that KC-based attacks observe leakage for valid ciphertexts, and thus can only target leakage from the polynomial multiplication operation of the secret key (Line 3 in the CPAPKE.Dec procedure in Alg. 6) for key recovery [PPM17, PP19, YRZ+23, MWK+22]. However, CC-based side-channel attacks form the largest category of attacks on Kyber KEM, and are more powerful than KC attacks, as the attacker can exploit leakage from the entire decapsulation procedure for key recovery. Thus, protection against CC-based attacks is naturally costlier compared to KC-based attacks, and analyzing countermeasures against CC-based side-channel attacks remain the main focus of our work.

There exist concrete proposals for masking as well as combined masking and shuffling to protect against such CC-based attacks. However, they are typically expensive, incurring several factors of overhead, particularly in terms of runtime [BGR+21, HKL+22]. Thus, an alternative approach towards countering such attacks has been towards development of low-cost countermeasures that attempt to detect such malicious ciphertexts [XPRO20, RCDB22]. If detected as malicious, the DUT can choose to refresh the secret key, ensuring further exposure for attacks is prevented.

2.4 Detection Countermeasures against CC based SCA

In this section, we broadly discuss two concrete proposals for such detection based countermeasures against CC-based side-channel attacks.

Ciphertext Sanity Check: The malicious ciphertexts used for PC oracle and FD oracle-based attacks are typically very sparse and skewed, with most of the coefficients having a value of 0 [XPRO20, RCDB22]. However, valid ciphertexts are LWE instances whose coefficients are uniformly distributed in the range $[0, q]$. Thus, [RCDB22] proposed to test the entropy of input ciphertexts to detect PC oracle or FD oracle-based attacks. This countermeasures works in the following manner.

In the ciphertext sanity check, the mean and variance is used as measures for the entropy of the input ciphertext. We denote the mean and standard deviation of the coefficients of a polynomial $x \in R_q$ as $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ respectively. The main idea of the countermeasure is then to compute $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ for all the polynomials in the input ciphertext (i.e., $\mathbf{c}_1$ and c_2) and reject the ciphertext if the variance of these inputs is too low. One main advantage

of this approach is that the malicious ciphertexts are rejected even before manipulated by the decapsulation procedure. Thus, the attacker cannot observe any leakage corresponding to these malicious ciphertexts thereby offering concrete protection against PC oracle and FD oracle attacks that utilize skewed ciphertexts. Ravi *et al.* [RCDB22] showed that this countermeasure only incurs about 5% overhead in runtime. However, we remark that this countermeasure does not protect against the DF oracle based attacks [BDH+21], which already utilize uniformly random ciphertexts that are invalid.

Decapsulation Failure Check: A basic observation of all CC-based side-channel attacks is that all these attacks utilize invalid ciphertexts, that always induce a decapsulation failure. Thus, a more simpler countermeasure at the protocol level, would be to simply refresh the secret key immediately upon observing a decapsulation failure. In this case the attacker is restricted to recovering the secret key with only a single trace. This countermeasure therefore offers concrete protection against all types of CC based side-channel attacks, since most if not all CC based side-channel attacks require multiple traces (few tens to few thousands) for key recovery. In this work, we perform a concrete analysis of both these countermeasures and demonstrate novel CC based side-channel attacks that are capable of breaking both these countermeasures.

3 Analysis of Ciphertext Sanity Check Countermeasure

In this section, we will demonstrate a novel attack technique that enables the PC oracle and FD oracle-based attacks, which are originally not possible on implementations protected with the ciphetext sanity check countermeasure, with only a small reduction in the accuracy of the attack. This is done by masking a malicious attack ciphertext to look uniformly random using the public key. The setup for our attack is exactly the same as that of standard PC oracle and FD oracle-based SCA [RRCB20, XPRO20, RRD+23], and our technique simply involves adding a mask to the chosen ciphertexts used for the PC and FD oracle-based SCA. We start with briefly describing the adversary model applicable to our attack.

Adversary model: The attacker attempts to recover the long-term secret key sk used by the target's decapsulation procedure of Kyber KEM. We assume physical access to DUT performing decapsulation for power/EM measurements. Since our attack is a CC-based attack, we assume the attacker's ability to communicate with the target decapsulation procedure with chosen ciphertexts of their choice. We note that this is a standard adversarial model used in several CC-based side-channel attacks [RRCB20, XPRO20, BDH+21].

Attack Intuition: For the intuitive explanation, we will assume that the public key and ciphertext vectors only consist of one polynomial (i.e., $k = 1$) and that no ciphertext compression is used. The main idea behind the masking can be intuitively explained as follows: imagine a public key $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$, and an attack ciphertext $(\mathbf{u}_{atk}, v_{atk})$ used in a PC oracle or FD oracle attack, that would not pass the ciphertext sanity check. The goal is to obtain the message polynomial:

$$\Delta = v_{atk} - \mathbf{u}_{atk} \cdot \mathbf{s}. \tag{3}$$

Table 1: Attack parameters used for the CC-based side channel attack using skewed ciphertexts in [RRD+23].

$s[0][0] =$		-2	-1	0	1	2
$\mathbf{u}_{atk}$	v_{atk}			m		
208	624	1	0	0	0	0
208	832	1	1	0	0	0
208	1040	1	1	1	0	0
208	1248	1	1	1	1	0

To this end we adapt our attack ciphertext to $(\mathbf{u}_{atk} + \mathbf{A}, v_{atk} + \mathbf{b})$. The attack ciphertext looks uniformly random and is accepted. The message in this case is:

$$\begin{aligned}\Delta &= v - \mathbf{u} \cdot \mathbf{s} && (4)\\ &= v_{atk} + \mathbf{b} - (\mathbf{u}_{atk} + \mathbf{A}) \cdot \mathbf{s} && (5)\\ &= v_{atk} + \mathbf{A} \cdot \mathbf{s} + \mathbf{e} - \mathbf{u}_{atk} \cdot \mathbf{s} - \mathbf{A} \cdot \mathbf{s} && (6)\\ &= v_{atk} - \mathbf{u}_{atk} \cdot \mathbf{s} + \mathbf{e}. && (7)\end{aligned}$$

which is the original equation up to the small error e.

One countermeasure against this attack might be to check if the ciphertext is close to the public key, or more specifically whether $(\mathbf{u} - \mathbf{A}, v - \mathbf{b})$ is small. However, such a countermeasure can easily be circumvented by instead of adding the public key, adding a small multiple of the public key (e.g. $(-\mathbf{A}, -\mathbf{b})$ or $(2\mathbf{A}, 2\mathbf{b})$) or a rotated version of the public key (e.g. $(X \cdot \mathbf{A}, X \cdot \mathbf{b})$).

Attack on Kyber: We now go into more detail on the practical implementation of our attack, where we show how to mask the ciphertexts used for the PC oracle-based side-channel attack. As in previous works, the goal of our attack is to input a chosen ciphertext $(\mathbf{u}_{atk}, v_{atk})$ of the form $\mathbf{u}_{atk} = [y, 0, 0]$ and $v_{atk} = z$ with $|y| < q/4$.

The first message bit $m[0]$ is then calculated as

$$m[0] = \left\lfloor \frac{2}{q} \cdot (z - \mathbf{s}[0][0] \cdot y) \right\rceil \quad (8)$$

where $m[0]$ is therefore only dependent on the first coefficients of the secret (i.e.) $\mathbf{s}[0][0]$, while all other message bits are zero. The message thus only has two possible values, and the attacker can instantiate a practical PC oracle through side-channel leakage from the decapsulation procedure. The attacker chooses similar values for the tuple (y, z) such that the message can be used as a binary distinguisher for the first secret coefficient.

By repeating this procedure for other values of y and z, and other coefficients of $\mathbf{s}$, one can retrieve the full secret key. Notice that this is a well-known attack procedure [RRCB20,RRD+23]. Table 1 shows the values of y and z used by Rajendran et al. [RRD+23] to mount an attack against Kyber768.

The attack strategy described above does not pass the ciphertext sanity check, and thus we will have to add the public key to mask the input ciphertext. A first practical problem is that the size of $\mathbf{u} \in R_q^{k \times 1}$ is not equal the size of $\mathbf{A} \in R_q^{k \times k}$, and the same holds for the values $v \in R_q^{1 \times 1}$ and $\mathbf{b} \in R_q^{k \times 1}$. This can be solved by selecting only one row of the matrix $\mathbf{A}$, which we will indicate with $\mathbf{a}^*$, and the corresponding polynomial element of the vector $\mathbf{b}$, which will be indicated with b^*. The introduced error will be indicated e^* and is the corresponding element of the vector $\mathbf{e}$

A second inconvenience is that the ciphertext is compressed, which means we cannot exactly input the ciphertext. This introduces an error to the values of $\mathbf{a}^* + \mathbf{u}_{atk}$ and $b^* + v_{atk}$ that can be inputted in the attack ciphertext. One can calculate that this results

in a message Δ defined as:

$$\Delta = v_{atk} - \mathbf{u}_{atk} \cdot \mathbf{s} + E, \quad \text{where } E = \mathbf{e} - \mathbf{s}^T \cdot \Delta\mathbf{u} + \Delta v.$$

In the above equation, the value of $\Delta\mathbf{u}$ and Δv implicitly captures the additional rounding error that occurs due to addition of the public key mask. This is essentially the noisy message equation from 2.2.1 where $\mathbf{r} = 1$, $\mathbf{e}_1 = 0$ and $e_2 = 0$ (interestingly, one can see the original unmasked attack as an instance where $\mathbf{r} = 0$). This means that the message Δ calculated in the attack (see 7), has an additional error term due to the compression and decompression.

For power-of-two moduli schemes (e.g., Frodo, Saber), there is typically an additional public key compression of the $\mathbf{b}$ term. The compression, in this case, works as follows: first LSBs are discarded from $\log_2(q)$ bits to $\log_2(p)$ bits in public key compression, then from $\log_2(p)$ to $\log_2(T)$ in ciphertext compression. This means that the public key compression is essentially contained in the ciphertext compression, so its effect can be ignored.

For a given mask $(\mathbf{a}^*, b^*)$, the value of E is nearly identical for the recovery of different coefficients. During an attack, Δv and only one coefficient of Δu are changed slightly due to different rounding while all other coefficients are unchanged. Thus, value of E is only changed slightly, which is experimentally verified. This means that each mask $(\mathbf{a}^*, b^*)$ is tied to a certain unknown approximate error E^*. If this approximate error is too large, the attack will fail with high probability, but if this approximate error is low enough, the attack proceeds without the masking having any effect other than defeating the detection countermeasure. Note that one cannot predict the value of this error E^* in advance. If the key recovery fails, one can restart the attack with a different mask (e.g. using a different row of $\mathbf{A}$ and $\mathbf{b}$, using a rotated version $(X \cdot \mathbf{a}^*, X \cdot \mathbf{b}^*)$ or using a multiple of the mask $(c \cdot \mathbf{a}^*, c \cdot \mathbf{b}^*)$).

Result: We performed software simulations of our proposed ciphertext masking approach on the malicious chosen ciphertexts of the binary PC oracle attack proposed in [RRD+23]. We did not perform side-channel evaluation, as the realization of the oracle has already been demonstrated in [RRD+23], and it applies in exactly the same manner for our proposed attack. In our experiments on Kyber768 to recover 200 secret keys, we were able to recover all of them with 100% success rate. However, a randomly selected mask $(\mathbf{a}^*, b^*)$ yields a success probability of 57.8% for full key recovery. Since the success of each run is independent, the expected number of runs is $1/0.578 = 1.73$, which is the expected number of runs for full key recovery. This means that one has to restart only 0.73 times on average (i.e. $\sum_{i=1}^{\infty} i \cdot p \cdot (1-p)^i$) before hitting a working mask and thus performing a successful attack. Our proposed ciphertext masking approach can be used for the binary and parallel PC oracle attacks as in [RRD+23], thereby only requiring a few hundred ciphertext queries for full key recovery. Our technique can also be used to mask ciphertexts for the more efficient FD oracle-based side-channel attacks [XPRO20], which have been shown to require ≈ 10 traces for full key recovery.

Comparison with DF Oracle-based SCA [BDH+21, PP21, HPP21, Del22]: Apart from our proposed attack, we remark that DF oracle based side-channel attacks [BDH+21] can also defeat the ciphertext sanity check countermeasure, as they utilize uniformly random ciphertexts. However, DF oracle-based attacks [BDH+21, PP21, HPP21, Del22] are known to require a few thousand ciphertext queries for full key recovery ($\approx 6.5k$ to $7k$ ciphertext queries for Kyber768). However, we show that our simple masking technique can enable the more efficient PC oracle [RRCB20, RRD+23] and FD oracle based attacks [XPRO20, RBRC20] capable of extracting much more information about the secret key, thus requiring much lesser number of traces (few tens to few hundred chosen-ciphertext queries) compared to the DF oracle-based attacks [BDH+21, PP21, HPP21, Del22].

4 Analysis of Decapsulation Failure Check Countermeasure

The goal of this section is to develop attacks that can circumvent countermeasures that refresh the secret key when a decapsulation failure is observed. Defeating this countermeasure calls for a completely new approach that uses valid chosen ciphertexts for key recovery. This makes our approach significantly different than typical state-of-the-art attacks that rely on invalid ciphertexts which fail the re-encryption check with overwhelming probability. We start with briefly describing the adversary model for our attack.

Adversary Model: The standard adversarial model used for CC based side-channel attacks (described in Section 3) is applicable to this attack, along with some additional assumptions. All the chosen ciphertexts used in our attack are valid ciphertexts. Since it is a profiled attack, the attacker has access to a clone device, on which he/she can control the secret key, so as to build side-channel templates for any intermediate variable of his/her choice.

Targets for CC-based SCA with Valid Ciphertexts: Since the attacker needs to rely on valid ciphertexts to defeat the decapsulation failure check, they are limited to targeting operations in the decryption procedure for key recovery. KC-based side-channel attacks [PPM17, PP19, MWK+22, YRZ+23, BBB+23] which utilize valid ciphertexts work with valid ciphertexts, and thus can defeat the decapsulation failure check countermeasure in multiple ways, through single trace as well as multiple trace attacks. However, these attacks solely target the polynomial multiplication operation. Thus, protecting this operation alone can protect against key recovery when considering KC-based SCA. This raises the question whether there are other potential operations within the decryption procedure, that can be targeted with valid ciphertexts, bypassing the decapsulation failure check countermeasure.

In this work, we show a novel CC-based attack with valid ciphertexts capable of full key recovery by targeting a different operation (i.e.) manipulation of the noisy message polynomial m' within the decryption procedure. Exploiting this leakage for practical attacks is not trivial due to two main reasons. Firstly, it is not possible to mount CPA-style attacks that use a divide-and-conquer approach since every coefficient of m' depends on all coefficients of the secret key. Secondly, while the value of m' depends on the secret key, operations manipulating m' do not directly involve the secret key. This means that one will only obtain indirect information on the secret, and no direct information on individual coefficients as in typical CC attacks. Consequently, to obtain the key from the leakage of m', one needs an additional key recovery step. In this work, we demonstrate how to overcome these obstacles and show that an attacker who queries only valid chosen ciphertexts can still exploit leakage of m' for efficient key recovery attacks.

4.1 Intuitive Explanation of our Attack

We represent the noisy message polynomial m' as $m' = \lfloor q/2 \rceil \cdot m + \Delta m$ where Δm is the decryption noise component. We recall from Section 2.2.1, that Δm is linearly dependent on the secret $\mathbf{s}$ and error $\mathbf{e}$ as $\Delta m = \langle \mathbf{e}, \mathbf{r} \rangle - \langle \mathbf{s}, \mathbf{e}_1 + \Delta \mathbf{u} \rangle + e_2 + \Delta v$.

For a valid chosen ciphertext (i.e.) ciphertext generated by running the encapsulation procedure (Alg. 5), the attacker knows $\mathbf{r}, \mathbf{e}_1$ and e_2, as well as the extra noise factors $\Delta \mathbf{u}$ and Δv, which are caused by the compression and decompression of ciphertexts. Knowledge of $\Delta m \in R_q$ for $2 \cdot k$ ciphertexts results in direct recovery of the secret key $\mathbf{s}$ using linear algebra. The same applies if single coefficients of Δm can be recovered across $2 \cdot k \cdot n$ ciphertexts. We show that leakage of the Hamming weight of coefficients of m' for valid

ciphertexts can be used to define bounds on Δm. From there we can use solvers for systems of linear inequalities to retrieve the secret vectors $\mathbf{s}$ and error $\mathbf{e}$. Our chosen ciphertext attack proceeds in three steps:

Step 1. Recovering $\mathsf{HW}(m')$ through Side-Channels: During decapsulation, the processing of the polynomial $m' \in R_q$ leaks the Hamming weights (HW) of each of its coefficients. In Section 5, we experimentally demonstrate how to recover the HW of the coefficients of m' from different implementations of Kyber.

Step 2. Relating $\mathsf{HW}(m')$ with Δm: We observe that for a message bit $m_i = 0$, the distribution of $\mathsf{HW}(m'[i])$ has a very clear bias, that clearly indicates the sign of $\Delta m[i]$. In particular, when $\mathsf{HW}(m'[i])$ is very low, then $\Delta m[i] > 0$ with a very high probability, while a very high value for $\mathsf{HW}(m'[i])$, signifies $\Delta m[i] < 0$ with a very high probability. This information can be used to construct a linear inequality in the coefficients of $\mathbf{s}$ and $\mathbf{e}$. Generating a sufficient number of inequalities can be used to recover the secret key.

Step 3. Solving Linear Inequalities in s and e for Key Recovery: In this step we solve the linear inequalities from Step 2 to retrieve the secret vectors $\mathbf{s}$ and $\mathbf{e}$. This approach is shown in Section 6. Pessl and Prokop [PP21], Hermelink *et al.* [HPP21] and later Delvaux [Del22] proposed efficient methods based on the Belief Propagation techniques to solve these linear inequalities in the context of fault attacks. We propose a novel greedy solver approach that is simpler to understand, uses lesser compute resources and at the same time requires 2× fewer inequalities to extract the secret key.

In Section 5, we will provide details of Step 1 of our attack, involving recovery of $\mathsf{HW}(m')$ through side-channels. Subsequently, in Section 6, we will demonstrate how this side-channel information can be fed into our novel solver for key recovery (Steps 2 and 3 of our attack).

5 Recovering $\mathsf{HW}(m')$ through Side-Channels

We start by describing the experimental setup used in this work. We performed experiments on reference and optimized implementations of Kyber KEM from the well-known *pqm4* library [KRSS] running on the STM32F407 microcontroller clocked at 24 MHz. We obtain EM side-channel leakage using a Langer RF-U 5-2 near-field EM probe, and the traces were collected using a Lecroy 610Zi oscilloscope at a sampling rate of 1.25 GSam/sec, amplified 30dB with a pre-amplifier and filtered using a 48 MHz low-pass filter.

5.1 Detecting Leakage from Coefficients of m'

We now demonstrate leakage detection from manipulation of single coefficients of m' in two different implementations of Kyber KEM: (1) Reference Implementation [ABD+20] (2) Assembly-Optimized Implementation [HZZ+22].

5.1.1 Reference Implementation

Refer to Fig.1 for the assembly code snippet of the subtraction operation between $\mathbf{s}^T \cdot \mathbf{u}$ and v (Line 4 in CPAPKE.Dec procedure in Alg. 3), when compiled with the O3 optimization level. The coefficients of m' are computed and stored in memory, sequentially one coefficient at a time, which allows to observe leakage from multiple coefficients simultaneously. To illustrate this, we simultaneously profile leakage of the first 8 coefficients $m'[i]$ for $i \in \{0, \ldots, 7\}$.

We build $100k$ valid ciphertexts for random messages m, but choose $m_i = 0$ for $i \in \{0, \ldots, 7\}$. We perform CPA for the coefficients $m'[i]$ for $i \in \{0, \ldots, 7\}$ and refer to

Fig. 3(a) that shows 8 CPA plots, one for each coefficient $m'[i]$ for $i \in \{0, \ldots, 7\}$ for the O3 optimized implementation. The peaks for each of the coefficients are easy to distinguish and sufficiently separated from one another indicating clear leakage from individual coefficients.

5.1.2 Assembly Optimized Implementation

We studied the assembly-optimized implementation of Kyber KEM from Huang *et al.* [HZZ+22], and refer to Fig.2 for the highly optimized hand-written assembly code snippet of the polynomial subtraction operation. We observe that 10 coefficients of its operands are simultaneously loaded into 5 registers using the vectorized ldmia instruction (Lines 3, 5). Each register is packed with two coefficients (upper and lower halves), and the subtraction operation for the 10 coefficients is carried out by five usub16 instructions (Line 7-11). Then, the vectorized store instruction (stmia) is used to simultaneously store 10 coefficients in memory. Thus, one can only observe simultaneous leakage from 10 coefficients, which results in significant noise when targeting leakage of one coefficient.

Thus, to detect leakage from a single coefficient say $m'[0]$, we adopt the following noise reduction technique. From Fig.4(a)-(b), we observe that the HW distribution of coefficients when $m_i = 1$ is much smaller compared to when $m_i = 0$. Thus, to reduce noise from the other coefficients $m'[i]$ for $i \in \{1, \ldots, 9\}$, that are stored along with $m'[0]$, we fix their corresponding message bits $m_i = 1$ for $i \in \{1, \ldots, 9\}$ in the encapsulation procedure. This approach can be followed to exploit simultaneous leakage from multiple coefficients in the following manner. To exploit leakage from say 13 coefficients $m'[i \cdot 10]$ for $i \in [0, 12]$, we choose $m_{(i \cdot 10)} = 0$ for $i \in [0, 12]$, while the bits corresponding to the remaining coefficients that are simultaneously stored along with this $m'[i \cdot 10]$ for $i \in [0, 12]$ are fixed to 1. The remaining bits of the message m are randomized. We build $100k$ ciphertexts for such random messages.

Refer Figure 3(b) shows 13 CPA plots, corresponding to leakage from the 13 coefficients corresponding to $m'[i \cdot 10]$ for i in $\{0, \ldots, 12\}$. Notice that, even though 10 coefficients are simultaneously stored in memory, we are able to observe leakage from a single targeted coefficient by setting the other 9 message bits to 1. We can observe the CPA peaks corresponding to the targeted coefficients, and that the peaks are identical and uniformly separated from one another. This demonstrates that simultaneous leakage from one every ten coefficients can be exploited for key recovery. We have thus shown that leakage from single coefficients of m' can be observed in both the reference and assembly optimized implementations. In the following, we demonstrate how the detected leakage can be exploited to recover HW$(m'[i])$.

```
1
2  /* Load Coeff. of A (2 bytes) in r2 */
3  ldrh.w r2, [r4, #2]!
4  /* Load Coeff. of B (2 bytes) in r1 */
5  ldrh.w r1, [r3, #2]!
6  /* Check Loop Counter for Iteration */
7  cmp    r4, r5
8  /* Subtract A from B and store in r2 */
9  sub.w  r2, r2, r1
10 /* Store result in r2 into memory */
11 strh.w r2, [r0, #2]!
12 /* Branch if not end of loop */
13 bne.n  0x8005d30
```

Figure 1: Assembly code snippet of PolySub operation (polynomial subtraction) when compiled with O3 optimized implementation. The target store operation that leaks the HW of the coefficients is highlighted in red.

```
1
2  /* Load 10 Coeffs. of A in r3 - r7 */
3  ldmia  r1!, {r3, r4, r5, r6, r7}
4  /* Load 5 Coeffs. of B in r8 - r12 */
5  ldmia.w r2!, {r8, r9, r10, r11, r12}
6  /* Simultaneously subtract two coeffs in r3-r8 from r8-r12 */
7  usub16 r3, r3, r8
8  usub16 r4, r4, r9
9  usub16 r5, r5, r10
10 usub16 r6, r6, r11
11 usub16 r7, r7, r12
12 stmia r0!, r3, r4, r5, r6, r7
13 /* Update loop counter */
14 subs.w lr, lr, #1
15 /* Branch if not end of loop */
16 bne.w  0x8005a14
```

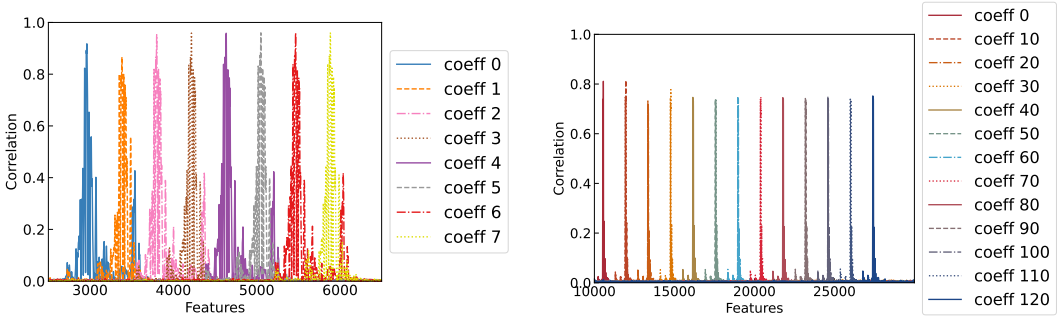
Figure 2: Assembly code snippet of PolySub operation (polynomial subtraction) for the highly optimized assembly implementation of Kyber KEM for the ARM Cortex-M4 microcontroller. The target store operation that leaks the HW of the coefficients are highlighted in red.

5.2 Building a Classifier for HW($m'[i]$)

The HW classifier is built in two phases: (1) Profiling phase and (2) Recovery phase. We illustrate the technique for a single coefficient $m'[0]$ and the same can be repeated for other coefficients.

5.2.1 Profiling Phase

The profiling phase is intended to build templates for the HW of the individual coefficients of the noisy message polynomial m'.This phase is carried out using leakage from a clone device. From the CPA plot corresponding to $m'[0]$ (Fig.3), we select those features whose correlation value is above a certain threshold Th_0, as our points of interest (PoI) denoted as $\mathcal{P}_0$. We stress that Th_0 is a parameter of the experimental setup, and can be experimentally determined. We use the selected features $\mathcal{P}_0$ to build a reduced trace set $\mathcal{RT}_{(0,i)}$ for every possible value of HW($m'[0]$) (i.e.) HW of the message polynomial coefficient $m'[0]$ that lies in the range $[0, 16]$. We can compute the mean and co-variance matrix of each reduced trace set $\mathcal{RT}_{(0,i)}$ for $i \in \{0, \ldots, 16\}$, which we denote as $\mu_{(0,i)} \in \mathbb{R}^{\|\mathcal{P}_0\|}$ and $\Sigma_{(0,i)} \in \mathbb{R}^{(\|\mathcal{P}_0\|)\times(\|\mathcal{P}_0\|)}$. Thus, the reduced template for HW($m'[0]$) $= i$ is denoted as $tmp_{(0,i)} = (\mu_{(0,i)}, \Sigma_{(0,i)})$ for $i = \{0, 1\}$. The same procedure can be repeated to build



(a) Ref. Implementation (O3-ref) (b) Assembly Optimized Implementation

Figure 3: Correlation Power Analysis (CPA) plot corresponding to coefficients of the noisy message polynomial m' for the reference implementation O0-optimized (a), O3-optimized optimization (b) and Assembly optimized implementation (c).

side-channel templates for multiple coefficients of m'.

To perform classification of the HW classes, we have decided to Random Forest, the reason being is that it has been shown to be successful in the previous works [CDCG22] and it does not have to deal with a more complex hyperparameter tuning usually encountered in more complex models. Random Forest or RF [Bre01] is an ensemble learning algorithm, based on the construction of multiple decision trees. The predictions from these decision trees are combined (for example, through majority voting) in order to achieve a better prediction. The individual decision tree by itself is usually sensitive to small changes in the training data, and when grown large enough, will usually tend to overfit the training data. RF is designed to address these problems of instability in the decision tree, and multiple trees are allowed to grow large, without the need of post-processing. These decision trees are trained on different subsets of the training data using bootstrapping methods (the training data is sampled uniformly with replacement), and the decision is then made by taking the average or majority voting of the decisions given by the trees.

5.2.2 Recovery Phase

In the recovery phase, we obtain a trace tr for a chosen ciphertext from the decryption procedure, and build a reduced trace tr'_0, corresponding to $\mathcal{P}_0$. We utilize the trained RF model for prediction, with the number of trees fixed to $1.5k$, where we observed a good accuracy performance on the validation set. To check if the data imbalance affects the performance, we also verify other statistical tests, such as precision and recall, and obtain a similar score as the accuracy. We then fit in the attack traces to obtain the prediction, which will then be forwarded to the solver to recover the key.

5.2.3 Experimental Results for HW Recovery

We performed experimental validation to recover the HW of single coefficients of m' for both the reference implementation and assembly optimized implementations of Kyber KEM. For the reference implementation compiled with `O3` optimization level, we obtain an accuracy of 91.1% in recovering the HW of single targeted coefficients. In case of the assembly optimized implementation, we obtain a much lower accuracy of 32.0% for recovering HW of single targeted coefficients. This much lower accuracy for the HW classifier can be attributed to the noise due to simultaneous storage of multiple coefficients. However, in the following section, we show that such imperfect HW classifiers with low accuracy do not deter key recovery using valid chosen ciphertexts.

6 Key recovery

We have shown the attacker's ability to recover $\mathsf{HW}(m')$ (i.e.) HW of individual coefficients of m'. In this section, we demonstrate a key recovery attack exploiting this information. We start by explaining how $\mathsf{HW}(m')$ can be used to derive information about key-dependent noise component Δm and thereby construct linear inequalities in the secret key coefficients. Subsequently, we explain our novel solver to perform full key recovery.

6.1 Recovering Information on Δm using $\mathsf{HW}(m')$

In this section, we consider the type of information we can learn about the decryption noise Δm from the leaked Hamming weight of the message $\mathsf{HW}(m')$. The intuitive idea of our approach is to characterize the relation between this Hamming weight and the sign of Δm. From this characterization, we can assign to each Hamming weight a probability of the sign being positive or negative.

6.1.1 A First Approach

It is known that the coefficients of m' are distributed as a narrow bell-shaped discrete distribution around $q/2$ or 0 in the following manner

$$m'[i] = \begin{cases} q/2 \pm \Delta m[i] & \text{for } m_i = 1, \\ 0 \pm \Delta m[i] & \text{for } m_i = 0, \end{cases} \tag{9}$$

where the standard deviation of the noise $\Delta m[i]$ is $\sigma(\Delta m[i]) \approx 79$.

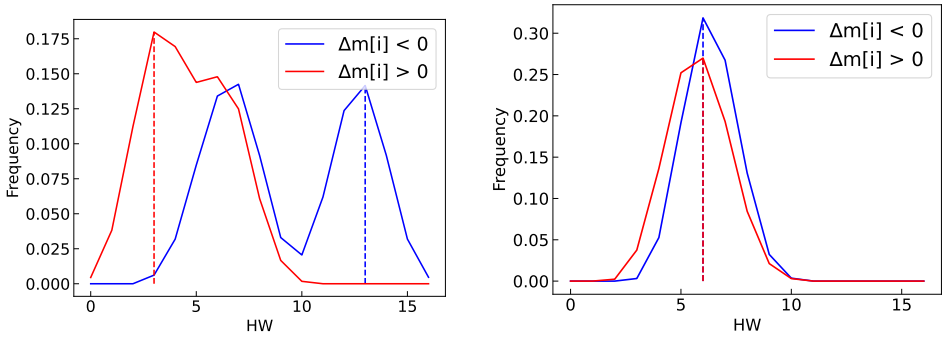
In most practical implementations of Kyber, such as *pqm4*, coefficients of m' are represented as signed 16-bit integers. Then, because of the narrow distribution of the noise $\Delta m[i]$, we hypothesized that it should be possible to learn something about the size of $\Delta m[i]$ from the observation of HW $(m'[i])$.

Figure 4 shows the distribution of $\mathsf{HW}(m'[i])$ when $\Delta m[i] \geq 0$ and $\Delta m[i] < 0$, considering $m[i] = 0$ and $m[i] = 1$. We can see that, when $m[i] = 0$, there is a clear distinction between positive and negative errors. This happens since a small negative error when added to a zero message bit (i.e. $m[i] \lfloor q/2 \rceil = 0$), results in a small negative number, which has a very high Hamming weight in the typically used two-complement representation. Similarly, small positive errors would result in values around 0 on the positive side, which in turn have very small Hamming weights. On the other hand, when $m[i] = 1$, then the HW distributions for positive and negative noise $\Delta m[i]$ are very close to each other, and therefore it is hard to obtain meaningful information on $\Delta m[i]$ from HW $(m'[i])$.

From this observation, if an attacker learns the HW of multiple $m'[i]$ where $m[i] = 0$, they can build a series of inequalities on $\Delta m[i]$. These types of inequalities can then be solved using Belief Propagation algorithms [PP21, Del22, HPP21]. While this approach is theoretically sound, it wastes some of the information that comes from the Hamming weight. In the following section, we discuss how to obtain more information than only the sign of $\Delta m[i]$ from HW $(m'[i])$.

6.1.2 Obtaining Tighter Inequalities

The Hamming weight of the message can not only be used to infer information about the sign of Δm but also on the possible values that Δm can take. That is, a certain Hamming weight can only be linked to a limited number of possible values of Δm and some Hamming weights can only occur within a limited range of values. Therefore, it is possible to compute both a lower bound and an upper bound on $m'[i]$ based on the Hamming weight. There is one important caveat when implementing this idea in practice:



(a) Message Bit $m_i = 0$ (b) Message Bit $m_i = 1$

Figure 4: Distribution of HW of message polynomial coefficients $m'[i]$ for positive error $(\Delta m[i] > 0)$ and negative error $(\Delta m[i] < 0)$, when $m_i = 0$ (a) and $m_i = 1$ (b).

Table 2: Maximum and minimum values of $m'[i]$ for each possible value of its Hamming weight that was observed in 100,000 decryptions using Kyber's reference implementation.

$\mathrm{HW}(m'[i])$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\max m'[i]$	0	256	320	328	338	351	324	319	316	251	253	-32	-16	-8	-4	-2	-1
$\min m'[i]$	0	1	-257	-256	-254	-250	-352	-357	-314	-338	-319	-332	-324	-305	-321	-257	-1
Spread	0	255	577	584	592	601	676	676	630	589	572	586	308	297	317	255	0

efficient Kyber implementations perform lazy modular reduction, and therefore it is not guaranteed that $-q/4 \leq m'[i] < q/4$ before the reduction. While it is not easy to give a theoretical estimate on the bounds on $m'[i]$ for each Hamming weight from 0 to 16, we can run a simulation to compute these values.

Table 2 shows the maximum and minimum values of $m'[i]$ observed for each possible Hamming weight, considering Kyber's reference implementation. The tightness of the inequalities are represented by the spread, which is simply the difference between $\max m'[i] - \min m'[i]$. It is also possible to see the effect of the lazy reduction. Notice how one would expect $\max m'[i]$ to be -64 for the case when $\mathrm{HW}(m'[i]) = 10$, since it's two's complement representation `0b1111111111000000` makes it the maximum value that can be the result of a narrow distribution around 0 with the given Hamming weight. However, instead we observe 253 because $\mathrm{HW}(253 + q) = \mathrm{HW}(\texttt{0b110111111110}) = 10$.

Now, for each observed $HW(m'[i])$ with $m[i] = 0$, an attacker is able to construct two inequalities, one for the maximum and one for the minimum. This gives more information to the inequalities solver, which we will cover next.

6.2 Solving Inequalities to Find The Secret Key

In this section, we formalize how a system of linear inequalities related to the secret key is built with SCA information. We then construct a new solver to retrieve the secret key from these inequalities.

6.2.1 Defining the System of Linear Inequalities Related to the Key

In the previous section, we saw that when $m[i] = 0$, then $m'[i] = \Delta m[i]$, and $\mathrm{HW}(m'[i])$ gives us information on $\Delta m[i]$. Suppose an attacker asks the target device to decrypt a number κ of ciphertexts, whose corresponding messages are $m_1, \ldots, m_\kappa$. Similarly, let $\mathbf{r}_j, (\mathbf{e}_1)_j, \mathbf{u}_j, (e_2)_j$ and v_j be the values resulting from the encryption of the corresponding messages m_j. Therefore, notice that these resulting values are known by the attacker.

Now, for any indexes $(j, i) \in \{1, \ldots, \kappa\} \times \{0, \ldots, 255\}$, we know that

$$\Delta m[i] = \langle \mathbf{r}_j, \mathbf{e} \rangle [i] - \left\langle (\mathbf{e}_1)_j + \Delta \mathbf{u}_j, \mathbf{s} \right\rangle [i] + (e_2)_j [i] + \Delta v_j[i].$$

But, from Equation 1, we have

$$\begin{cases} \langle \mathbf{r}_j, \mathbf{e} \rangle [i] = \langle \texttt{negashift}_i (\mathbf{r}_j), \texttt{poly\_to\_vec} (\mathbf{e}) \rangle, \\ \left\langle (\mathbf{e}_1)_j + \Delta \mathbf{u}_j, \mathbf{s} \right\rangle [i] = \left\langle \texttt{negashift}_i \left((\mathbf{e}_1)_j + \Delta \mathbf{u}_j \right), \texttt{poly\_to\_vec} (\mathbf{s}) \right\rangle. \end{cases}$$

Therefore, if we let

$$\mathbf{h}_{(i,j)} = \left[\texttt{negashift}_i (\mathbf{r}_j) \parallel - \texttt{negashift}_i \left((\mathbf{e}_1)_j + \Delta \mathbf{u}_j \right) \right] \in \mathbb{Z}_q^{2kn}, \tag{10}$$

then

$$\Delta m_j[i] = \left\langle \mathbf{h}_{(i,j)}, [\, \texttt{poly\_to\_vec} (\mathbf{e}) \parallel \texttt{poly\_to\_vec} (\mathbf{s}) \,] \right\rangle + (e_2)_j [i] + \Delta v_j[i].$$

Algorithm 7: Greedy key recovery algorithm.

Input: $\mathbf{H} \in \mathbb{Z}^{2\gamma \times 2kn}$, $\mathbf{w} \in \mathbb{Z}^{2\gamma}$
$\mathbf{x} \leftarrow \mathbf{0} \in \mathbb{Z}^{2kn}$
for $\mathtt{it} = 1$ **to** `max_iterations` **do**
  `action_scores` $\leftarrow$ ▷ dictionary where keys are actions and values are scores
  for $(j, v) \in \{0, \ldots, 2kn-1\} \times \{-\eta, \ldots, \eta\}$ **do**
    `action_scores`$[j, v] \leftarrow$ ComputeScore $(\mathbf{x}, (j, v), \mathbf{H}, \mathbf{v})$
  `best_actions` $\leftarrow$ List of keys (j, v) with the $\alpha_{\mathtt{it}}$ highest scores in `action_scores`
  for (j, v_{discard}) *in* `best_actions` **do**
    $\hat{v} \leftarrow \arg\max_v$ ComputeScore $(\mathbf{x}, (j, v), \mathbf{H}, \mathbf{w})$ ▷ Recompute current best v for index j
    $\mathbf{x}[j] \leftarrow \mathbf{x}[j] + \hat{v}$ ▷ Apply action $(j, \hat{v})$ to $\mathbf{x}$
$\mathbf{x}$

When $m_j[i] = 0$, then the attacker can use Table 2 to obtain two values $\omega_{(i,j)}$ and $\Omega_{(i,j)}$ such that $\omega_{(i,j)} \leq \Delta m_j[i] \leq \Omega_{(i,j)}$. This means that, in this case, the attacker can build the following two inequalities:

$$\begin{cases} \mathbf{h}_{(i,j)} \left[\mathtt{poly\_to\_vec}\left(\mathbf{e}\right) \| \mathtt{poly\_to\_vec}\left(\mathbf{s}\right) \right] + \left(e_2\right)_j [i] + \Delta v_j[i] - \omega_{(i,j)} & \geq 0, \\ -\mathbf{h}_{(i,j)} \left[\mathtt{poly\_to\_vec}\left(\mathbf{e}\right) \| \mathtt{poly\_to\_vec}\left(\mathbf{s}\right) \right] - \left(e_2\right)_j [i] - \Delta v_j[i] + \Omega_{(i,j)} & \geq 0. \end{cases} \tag{11}$$

The attacker can then select a number γ of pairs (j, i) such that $m_j[i] = 0$ and get a linear system of inequalities. This system of inequalities is represented by a matrix $\mathbf{H} \in \mathbb{Z}_q^{2\gamma \times 2kn}$, and a vector $\mathbf{w} \in \mathbb{Z}_q^{2\gamma}$ as defined next, based on Equation 11.

Each pair of rows of matrix $\mathbf{H}$ consists of the vectors $\mathbf{h}_{(i,j)}$ and $-\mathbf{h}_{(i,j)}$. Entries of $\mathbf{w}$ consist of the factors $\left(\left(e_2\right)_j [i] + \Delta v_j[i] - \omega_{(i,j)}\right)$ and $\left(-\left(e_2\right)_j [i] - \Delta v_j[i] + \Omega_{(i,j)}\right)$. Then, if γ is sufficiently large and if the inequalities are sufficiently tight, then a small solution $\mathbf{x}$ to the linear system $\mathbf{Hx} + \mathbf{w} \geq \mathbf{0}$ should give us the secret $\left[\mathtt{poly\_to\_vec}\left(\mathbf{e}\right) \| \mathtt{poly\_to\_vec}\left(\mathbf{s}\right) \right]$.

One may be tempted to use every single inequality that is generated. However, some of them will be very loose. There is an important trade-off when selecting a good spread value to filter pairs of inequalities. If it is too small, then a larger number of traces may be needed, but if it is too large, then the algorithm must be prepared to deal with a potentially very large number of inequalities. To find the spread minimizing the number of ciphertexts needed for a successful attack, we ran a simple simulation considering all spread values in Table 2, and obtained the value of $\mathtt{max\_spread} = 317$.

6.2.2 New Solver for Key Recovery

Until now, the most efficient solver available for recovering the key from inequalities was the Belief Propagation algorithm proposed by Pessl and Prokop [PP21] with the improvements by Delvaux [Del22]. This solver, however, has an important limitation: it is not efficient when dealing with lots of inequalities and requires a large amount of RAM.

We propose a simpler greedy algorithm to find the key from the inequalities constructed from the Hamming weight of the noisy message coefficients. Our algorithm is presented as Algorithm 7. It starts with $\mathbf{x} = \mathbf{0} \in \mathbb{Z}^{2kn}$ and then, on each iteration, it performs a number of actions that, according to the heuristic score function in Algorithm 8, should guide $\mathbf{x}$ closer to the solution. Each action is a pair (j, v), where j is an index of $\mathbf{x}$ and v is an integer that is added to $x[j]$. To obtain a faster convergence, we let the number $\alpha_{\mathtt{it}}$ of actions to be applied on iteration `it` to start as a large number and decay exponentially in the number of iterations.

The score function for actions defined in Algorithm 8 works as follows. First, it applies the action on $\mathbf{x}$ and verifies which inequalities are not satisfied after the action was applied. Then, for each unsatisfied inequality, it penalizes the action score with the numerical distance from the offending number to the closest value that satisfies the inequality.

Figure 5 shows a comparison between our greedy search solver and the one based on belief propagation (BP) [Del22]. We can see that, despite its simplicity, our solver

Algorithm 8: Score function.

Input: $\mathbf{x} \in \mathbb{Z}^{2kn}, (j, v) \in \{0, \ldots, 2kn-1\} \times \{-\eta, \ldots, \eta\}, \mathbf{H} \in \mathbb{Z}^{2\gamma \times 2kn}, \mathbf{w} \in \mathbb{Z}^{2\gamma}$

```
score ← 0
H_j ← jth column of H
t ← Hx + w + vH_j                    ▷ Resulting target vector when action (j, v) is applied
for i = 1 to 2γ do
    ▷ For each inequality, lower score by the distance it is from being satisfied
    if t[i] < 0 then
        score ← score − |t[i]|
score
```

compares favorably with respect to the number of ciphertexts needed to recover the key. Furthermore, it appears to be more robust with respect to noise in the SCA observations.

Notice that the performance of the belief propagation algorithm degrades fast, therefore we could not generate more points for larger standard deviations as the simulations take too long to finish. For comparison, when the SCA noise standard deviation is $\sigma = 0.5$, the BP algorithm takes, on average, more than 10 minutes, while our algorithm takes less than 20 seconds. We have summarized the complete attack methodology for our novel CC-based SCA using valid ciphertexts in Alg.9.

Algorithm 9: SCA-assisted CCA against Kyber using valid ciphertexts

Input: Target public key pk

```
▷ Offline ciphertext building
for j = 1 to max_ciphertexts do
    Run c_j ← Encaps(pk) keeping track of: m_j, r_j, (e_1)_j, (e_2)_j, u_j and v_j
    Compute the corresponding values of Δu_j and Δv_j
▷ Perform SCA on the decapsulation of valid ciphertexts
for j = 1 to max_ciphertexts do
    predicted_hws_j ← SCA on Decaps(c_j, sk) ∈ Z^256      ▷ Predicted HW of coefficients of m'
▷ Build approximate inequalities Hx + w ≳ 0 and find solution
Use the obtained values and Equations 10 and 11 to build (H, w)
[s'||e'] ← Solve(H, w)                                    ▷ Call Algorithm 7
if [s'||e'] is a valid secret key for pk then
    return [s'||e']
else
    ⊥
```

6.3 Experimental Results for Key Recovery

We implemented the entire attack on Kyber768 (recommended parameter set) to recover 10 random secret keys, and will separately explain the results for the reference (03 optimized) as well as assembly optimized implementation of Kyber KEM.

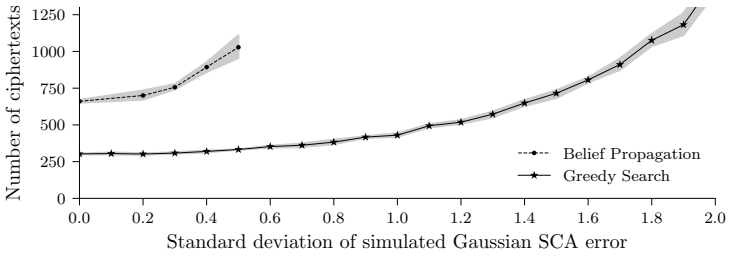


Figure 5: Comparison of our Greedy Search Solver compared to Belief Propagation for key recovery, considering different levels of Gaussian noise (i.e.) $\sigma = 0.0$ to 2.0.

Reference Implementation: We remark that any randomly generated message m contains an average of 128 zero bits, and thus the attacker can simultaneously exploit leakage from all the 128 bits. However, for our practical experiments, we only exploit leakage from the first 8 zero bits of the message for full key recovery, as it only involves construction of templates for 8 bits which is easier to handle compared to building and managing templates for 128 zero bits. This is just to simplify our experiments, and demonstrate a proof of concept for key recovery while only exploiting 8 zero message bits. We were able to recover all the 10 secret keys in about ≈ 5200 traces. We observed through experimental simulations that exploiting leakage from all zero bits in a randomly generated message (i.e.) average of 128 zero bits, enables full key recovery in just 325 traces, which correlates well with our attack simulations as presented in Figure 5.

Optimized Implementation: We showed in Section 5.1.2 that simultaneously storing 10 coefficients in memory ensures the attacker can only exploit leakage from one every 10 coefficient of m'. For illustration, we exploited leakage from 13 coefficients of m' (i.e.) m_{10*i} for $i \in \{0, \ldots, 12\}$, and were able to recover all secret keys with about ≈ 7800 traces. Thus, the number of traces is 24× higher than that to attack the reference implementation (325).

This can be attributed to the reduced number of exploited coefficients and the reduced accuracy of the HW classifier: $\approx 32\%$ for the optimized implementation, compared to $\approx 91\%$ for the reference implementation. While, intuitively, the low accuracy of 32% appears to be too low to allow key recovery, we notice that even if predictions are wrong but close to the actual HW, the inequalities generated may still be correct. Furthermore, if an inequality is wrong by a small margin, the score function ensures that the penalty is also small, thus still allowing the solver to converge.

In the optimized implementation, we exploited leakage of single coefficients from 13 vectorized store instructions. However, we remark that an attacker can exploit from several more vectorized store instructions, out of a total of 26 vectorized instructions for key recovery. Thus, it is clear that leakage of m' can be efficiently exploited by an attacker for key recovery with valid chosen ciphertexts. In the following, we demonstrate the applicability of our attack to the shuffling and masking countermeasures.

7 Attacking Shuffled Implementation

We now consider the case when the operations manipulating the coefficients of m' are shuffled. In this case, the attacker does not know when the zero message bits are being processed, and therefore even if they know that $m[i] = 0$, they cannot tell if $m'[i]$ have a high or low Hamming weight. But notice that, since the leakage of the message polynomial coefficients still exists, the attacker can recover HW of single coefficients of m' in the same manner as an unprotected implementation. This can be done by using templates created using leakage from a clone device, on which the attacker has knowledge of the random permutation used for every execution. However, we cannot directly use the same key recovery procedure without knowing the permutation used for processing these coefficients on the target device.

Attack Idea: Since the attacker can only exploit leakage from coefficients $m'[i]$ corresponding to $m_i = 0$, we propose to choose a message m that has only a very small number of zero bits. We remark that the attacker can explicitly choose the value of the message for valid chosen ciphertexts. Suppose that the chosen message m has $\theta = 2$ null bits at positions i and j, and that the attacker sees a sequence of Hamming weights $W = (w_1, w_2, \ldots, w_n)$ being processed. Then, if this sequence contains two very small values $w_{i^\star}, w_{j^\star} \leq 1$ or two very large values $w_{i^\star}, w_{j^\star} \geq 15$, then $i^\star$ and $j^\star$ must be associated with the zero positions i

and j in some way. Once this is done, we can proceed using the same key recovery strategy presented in the previous section, considering either 1 or 15 as the HW for both $m'[i]$ and $mt[j]$, depending on the HW values that were observed.

The main difficulty in using this idea in practice is that we need really tight intervals on a Hamming weight w to associate it to a zero message bit $m[i] = 0$, since we need to be sure to exclude all possible valid HW associated with $m[i] = 1$. This results in the observations within these intervals being very rare. Unfortunately, using $\theta = 2$ as in the example does only gives us $\binom{n}{2} = 32640$ different ciphertexts, which are not enough for us to make a high number of such rare observations. However, we can relax this idea and use messages with $\theta = 4$ null entries, but accept inequalities in case we see 2 extreme values. Then all indexes corresponding to $m[i] = 0$ be counted as having an extreme Hamming weight. Notice that this would generate some invalid inequalities.

As a proof of concept, we tested this idea and obtained a successful recovery, assuming perfect classification of the HW of the coefficients $m'[i]$. With 300000 inequalities, out of which around 16% were wrong, we were able to recover the key using a little more than 8 million ciphertexts. Although the number of ciphertexts is very large, it does support the validity of the approach, and we leave improving this value for future work.

8 Attacking Masked Implementation

In the masked implementation of the decryption procedure, the message polynomial m' is additively masked, and thus the attacker can observe leakage from the d independent arithmetic shares for each coefficient $m'[i]$ of the message polynomial, where $d-1$ is the order of masking. We performed our experiments on the first order masked implementation of Kyber768 from the open-source mkm4 library [HKL+22].

Consider the MaskedPolySub operation responsible for the masked computation of the subtraction $m' = v - \langle \mathbf{s}, \mathbf{u} \rangle$. Let $w = \langle \mathbf{s}, \mathbf{u} \rangle \in R_q$, and consider its arithmetic shares $w = w_0 + w_1 + \ldots + w_{d-1}$. The computation is done in two steps. First, it uses the poly_sub to compute $v - w_0$. Then the other shares are simply negated. Notice that poly_sub is similar to the unprotected version.

While the mkm4 library utilizes an assembly optimized implementation for the polynomial subtraction operation, for our analysis, we modified this routine with a C based implementation and compiled it with the `O0` compiler optimization level for simplicity. Our analysis of the assembly code also reveals that the attacker can observe leakage of the individual coefficients of all the arithmetic shares of m'. We performed CPA to detect leakage from coefficients of all the arithmetic shares of m' and refer to Figure 6 for the CPA plots, where we can observe leakage from coefficients of all shares of m'. For

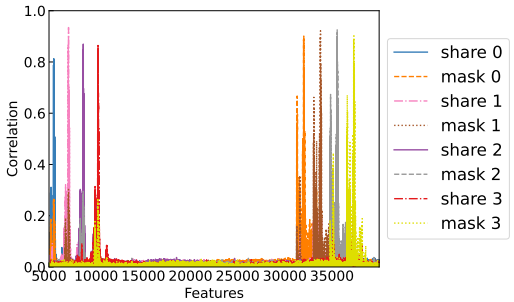


Figure 6: Correlation Power Analysis (CPA) plot for four coefficients of the two arithmetic shares of m', from the reference implementation of masked Kyber KEM, compiled with `O0` optimization level.

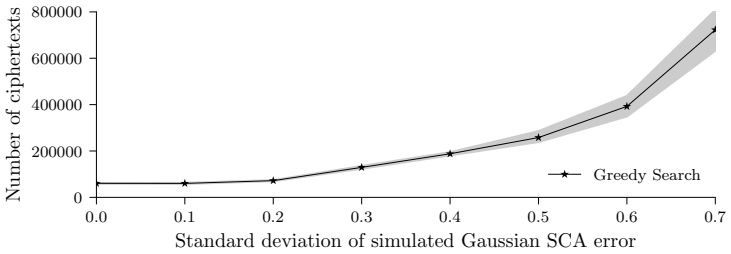


Figure 7: Number of ciphertexts required for key recovery considering considering different levels of Gaussian noise.

the classification of HW, we used the same technique that was used for the unprotected implementation and we were able to obtain an average accuracy of 94% for the HW classifier of all the targeted coefficients. We now show how the HW of coefficients of arithmetic shares of m' can be plugged into our novel greedy search algorithm for key recovery.

Key Recovery: From the key recovery point of view, the main difference when attacking the masked implementation is the construction of inequalities. Remember that, in the unprotected case, we used tables of minimums and maximum values of $m'[i]$ for each possible HW. This idea can be adapted for building inequalities for the masked implementation.

For the first order masked implementation, we build two 17×17 tables. In these tables, rows and columns represent the Hamming weights of the first and second shares of $m'[i]$. The entry with index (ω_0, ω_1) of the first table is the maximum observed value for $m'[i]$ when its shares have Hamming weights ω_0 and ω_1. The second table is similarly constructed, except that we take the minimum instead of maximum. To build the maximums and minimums tables, we used the code from *mkm4* implementation and observed 100000 decryptions. We remark that no SCA is needed for building this matrix, as they are only dependent on the implementation, not on the device. The resulting inequalities can then be directly plugged into the key recovery algorithm. Figure 7 shows the results performance of our key recovery algorithm in the masked case. Notice that we did not consider the Belief Propagation algorithm here because the required number of inequalities is too large for it to be efficient.

9 Conclusion

We performed the first security analysis of two detection-based SCA countermeasures against CC-based side-channel attacks: (1) Ciphertext Sanity Check and (2) Decapsulation Failure Check, demonstrating practical attacks against both countermeasures. We first report a novel attack to circumvent the ciphertext sanity checking, by simply applying the public key as a mask to a maliciously crafted ciphertext. We circumvent the decapsulation failure check, by proposing the first CC based side-channel attack that relies on valid ciphertexts for key recovery. Our attack exemplarily exploits leakage from the noisy message polynomial in the decryption procedure for full key recovery. We also introduced a simpler and improved inequality solver, that can recover the secret key with less than half the inequalities compared to these previous methods based on Belief Propagation.

We performed experimental validation of our attack on reference and optimized implementations of Kyber KEM on the STM32F4 microcontroller, requiring between $\approx 325-7800$ traces for full key recovery. We show how our attack can be adapted to both the shuffled and masked implementations, with appropriate increase in number of traces for key recovery. Our work therefore shows that low-cost detection countermeasures can be rendered completely ineffective, and do not offer standalone protection against CC-based side-channel

attacks. While these countermeasures are attractive for designers, our work encourages more study towards the development and analysis of new detection-based countermeasures against CC-based side-channel attacks.

Acknowledgment

Part of this work was discussed during the Dagstuhl Seminar-23152 titled "Secure and Efficient Post-Quantum Cryptography in Hardware and Software", Apr.2023 and we are thankful to Schloss Dagstuhl, its staff, organizers and participants of Dagstuhl Seminar-23152.

Jan-Pieter D'Anvers is funded by FWO (Research Foundation - Flanders) as junior postdoctoral fellow (contract number 133185). In addition, this work was supported by CyberSecurity Research Flanders with reference number VR20192203, the European Commission through the DIGITAL-SIMPLE project Be-QCI with contract number 101091625, and the Horizon 2020 research and innovation program Belfort ERC Advanced Grant 101020005.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the NIST post-quantum cryptography standardization process. Technical report, National Institute of Standards and Technology, 2022.

[ABD+20] Roberto Avanzi, Joppe W. Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber (version 3.0): Algorithm specifications and supporting documentation (October 1, 2020). *Submission to the NIST post-quantum project*, 2020.

[AHKS22] Amin Abdulrahman, Vincent Hwang, Matthias J Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. *Cryptology ePrint Archive*, 2022.

[BBB+23] Estuardo Alpirez Bock, Gustavo Banegas, Chris Brzuska, Łukasz Chmielewski, Kirthivaasan Puniamurthy, and Milan Šorf. Breaking dpa-protected kyber via the pair-pointwise multiplication. *Cryptology ePrint Archive*, 2023.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel van Beirendonck. Attacking and defending masked polynomial comparison for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):334–359, Jul. 2021.

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

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[CDCG22] Brice Colombier, Vlad-Florin Drăgoi, Pierre-Louis Cayrel, and Vincent Grosso. Profiled side-channel attack on cryptosystems based on the binary syndrome decoding problem. *IEEE Transactions on Information Forensics and Security*, 17:3407–3420, 2022.

[Del22] Jeroen Delvaux. Roulette: A diverse family of feasible fault attacks on masked Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 637–660, 2022.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, 2021.

[HKL+22] Daniel Heinz, Matthias J Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-order masked kyber on arm cortex-m4. *Cryptology ePrint Archive*, 2022.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on kyber. In *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*, pages 311–334. Springer, 2021.

[HZZ+22] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray CC Cheung, Çetin Kaya Koç, and Donglong Chen. Improved plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022.

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

[KRSS19] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. In *Second PQC Standardization Conference: University of California, Santa Barbara and co-located with Crypto 2019*, pages 1–22, 2019.

[MWK+22] Catinca Mujdei, Lennert Wouters, Angshuman Karmakar, Arthur Beckers, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Transactions on Embedded Computing Systems*, 2022.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *International Conference on Cryptology and Information Security in Latin America*, pages 130–149. Springer, 2019.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 37–60, 2021.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 513–533. Springer, 2017.

[RBRC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) NIST PQC candidates for practical message recovery and key recovery attacks. *IACR Cryptol. ePrint Arch.*, 2020:1559, 2020.

[RCDB22] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems*, 2022.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 123–146. Springer, 2020.

[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*, 2020(3):307–335, 2020.

[RRD+23] Gokulnath Rajendran, Prasanna Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. Pushing the limits of generic side-channel attacks on lwe-based kems-parallel pc oracle attacks on kyber kem and beyond. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 418–446, 2023.

[TUX+23] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 473–503, 2023.

[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*, pages 296–322, 2022.

[XPRO20] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, and David Oswald. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber. *IACR Cryptol. ePrint Arch.*, 2020:912, 2020.

[YRZ+23] Bolin Yang, Prasanna Ravi, Fan Zhang, Ao Shen, and Shivam Bhasin. Stamp-single trace attack on m-lwe pointwise multiplication in kyber. *Cryptology ePrint Archive*, 2023.