A Chosen-Ciphertext Side-Channel Attack on Shuffled CRYSTALS-Kyber

Hao Zhang, Zewen Ye, Teng Wang, Yuanming Zhang, Tianyu Wang, Chengxuan Wang, and Kejie Huang, *Senior Member, IEEE*

***Abstract*—The NIST Post-Quantum Cryptography (PQC) standardization has entered its fourth round, underscoring the critical importance of addressing side-channel attacks (SCA), a dominant threat in real-world cryptographic implementations, especially on embedded devices. This paper presents a novel chosen-ciphertext side-channel attack against CRYSTALS-Kyber (standardized as ML-KEM) implementations with Fisher-Yates shuffled polynomial reduction. We propose an efficient and fault-tolerant key recovery algorithm that, by crafting malicious ciphertexts, induces changes in the Hamming weight distribution of an intermediate polynomial's coefficients (the output of the shuffled polynomial reduction during decapsulation), enabling recovery of secret key coefficients from these changes. To ensure robustness, we propose an error-correction strategy that leverages the Hamming weight classifier's behavior to constrain and shrink the correction search space, maintaining effectiveness even with less accurate classifiers or in low-SNR environments. A Multi-Layer Perceptron (MLP) is employed for Hamming weight classification from side-channel traces, achieving 97.11% accuracy. We combine statistical analysis with explainable deep learning for precise trace segmentation during pre-processing. Experimental results demonstrate full key recovery with only an average of $10 + 354 \times 3$ ciphertext queries and a success rate of 97.98%, reducing the adversarial effort by 95.36% compared to contemporary bit-flip techniques. Although shuffling aims to disrupt temporal correlations, our results show that statistical features persist and leak through shuffled implementations. This work reveals enduring SCA risks in shuffled implementations and informs a broader reassessment of PQC side-channel resilience.**

***Index Terms*—Public-key cryptography post-quantum cryptography CRYSTALS-Kyber side-channel attack polynomial reduction shuffle.**

I. Introduction

THE security of modern public-key cryptosystems (e.g., RSA, ECC) relies on the computational hardness of mathematical problems like integer factorization and discrete logarithms. However, quantum computing advancements pose existential threats to these systems. For instance, Shor's algorithm [1] can solve integer factorization in polynomial time, thereby completely compromising RSA security. In response, the National Institute of Standards and Technology (NIST) initiated its Post-Quantum Cryptography (PQC) standardization project in 2016 to identify quantum-resistant algorithms.

In 2022, the National Institute of Standards and Technology formally standardized CRYSTALS-Kyber as the first post-quantum public-key encryption (PKE) and key encapsulation mechanism (KEM) [2], marking a critical milestone in the PQC standardization process. The Post-Quantum Cryptography standardization has now advanced to its fourth-round evaluation phase. A significant body of research has focused on developing high-performance hardware and software acceleration schemes for leading post-quantum cryptography candidates, including lattice-based cryptography [3]–[5], code-based cryptography [6], [7], and hash-based cryptography [8]. However, these acceleration efforts primarily target computational efficiency, with insufficient attention given to side-channel attack (SCA) protection. Although post-quantum cryptographic algorithms are mathematically secure under formal adversarial models (e.g., IND-CCA2 security), their practical implementations remain susceptible to side-channel attacks. This vulnerability arises from physical leakages, such as power consumption and electromagnetic emissions, during execution on hardware devices. Side-channel attacks represent one of the most prominent threats in real-world cryptographic deployments, especially in embedded systems. These systems are often deployed in resource-constrained environments like Internet of Things (IoT) devices and industrial controllers. They are particularly vulnerable due to their significant physical leakage profiles, which include observable power consumption and timing variations.

The core mechanism of side-channel attacks involves exploiting unintentional information leakage—such as timing variations, power consumption, electromagnetic emissions, or cache access patterns—generated during cryptographic operations. The field has evolved through several key milestones. Paul Kocher pioneered systematic research in 1996 by introducing timing attacks [9], which infer secret information through variations in encryption time. In 1999, Kocher et al. proposed Differential Power Analysis (DPA) [10], a method that correlates power consumption fluctuations with secret-dependent intermediate values. Subsequent research expanded the attack surface to include diverse strategies and leakage mediums: electromagnetic analysis [11] extracts secrets from EM radiation patterns; cache-timing attacks [12] exploit shared cache access latencies; and fault injection attacks [13] induce computational errors through physical interference. In recent years, deep learning techniques have significantly enhanced the analysis of high-noise side-channel data [14], while resource-constrained IoT devices have become frequent targets due to their inherent physical leakage vulnerabilities.

A. Related work

1) Protections: With the rising threat of machine learning-enhanced side-channel attacks on embedded systems,

The authors are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, 310027, China (e-mail: {floyd.haozhang, lucas.zw.ye}@zju.edu.cn, tengwang0318@gmail.com, {zhang_ym_b, wang_tianyu, wangchengxuan, huangkejie}@zju.edu.cn).

shuffling-based countermeasures have emerged as one of the critical defenses in securing Kyber implementations. [15] proposed multiple variant shuffling countermeasures of different complexity and levels of protection, aimed at protecting the Number Theoretic Transform (NTT), which incurs an additional 7%-78% overhead for Kyber. [16] introduced a hardware-friendly compact Fisher-Yates shuffling architecture that dynamically randomizes execution sequences while maintaining Kyber's parallel computation advantages with minimal additional resource overhead. These approaches highlight the trade-offs between shuffling complexity and security against advanced analytical attacks.

2) Attacks: Even though many side-channel protection strategies have been developed, they still cannot fully defend against the effects of side-channel attacks and fault injection attacks. Recent years have also witnessed a surge in machine learning-enhanced attacks against lattice-based cryptosystems such as Kyber and Saber, challenging common countermeasures. For lattice-based implementations employing shuffling defenses, several studies have demonstrated practical vulnerabilities.

Ngo et al. [17] employ bit-flip techniques [18] to extract Saber secret keys with $24\times257\times N$ traces ($N = 10$ repetitions per measurement). They construct specially chosen ciphertexts to induce bit-flips in the decrypted message and then use a Hamming weight classifier to recover the Hamming weights of the decrypted message from power traces collected during decryption, subsequently inferring decrypted message bits at flipped positions through Hamming weight variations. [19] recovers a Saber secret key using only 4608 traces by directly extracting the first and last Fisher-Yates shuffle indices from decryption power consumption to obtain the corresponding message bits. They then cyclically rotate the message via ciphertext modification and iteratively repeat trace collection and analysis to extract successive bits until the full message is recovered. This approach differs from prior work [20] that aimed to recover all shuffle indices directly from leakage; however, when the shuffle indices span 256 values, exhaustive index recovery becomes impractical due to its negligible success rate, limiting the effectiveness of direct index-recovery strategies.

Beyond message shuffling, [21] presents an adaptive belief propagation framework for decoding shuffling in NTT-based implementations. For fine-grained shuffling, shuffle nodes dynamically learn permutation rules by iteratively tuning exchange weights via KL-divergence minimization between belief propagation messages and physical leakage priors; for coarse-grained shuffling, two-point matching reconstructs operand sequences by correlating dual-node leakage (Load/Store Hamming weights) within butterfly units through Sinkhorn-Knopp-optimized probability matrices. This constitutes the first systematic toolkit for analyzing randomized NTT executions previously regarded as secure. However, any updates to shuffling permutation rules require complete relearning. [22] circumvents shuffling defenses through voltage glitch injection, then applies deep learning models based on Hamming weight leakage for key extraction.

Collectively, these results show that shuffling can be effectively compromised, particularly when augmented by machine learning.

B. Our Contributions

This work demonstrates an efficient and fault-tolerant chosen-ciphertext attack (CCA) enhanced by power side-channel analysis (SCA), targeting CRYSTALS-Kyber implementations protected with the Fisher-Yates shuffling countermeasure. We successfully recover long-term secret keys and expose vulnerabilities in shuffling. The contributions of this paper are as follows:

1) We combine statistical analysis (Correlation Power Analysis and Welch's t-test) and explainable deep learning techniques for the localization of points of interest (POI) and side-channel trace segmentation. Compared to using only statistical analysis, this hybrid approach achieves superior segmentation, capturing all side-channel leakage points and improving classifier accuracy. Compared to using only explainable deep learning techniques, POI localization is faster.
2) We design an efficient secret key recovery algorithm by leveraging statistical features from the side-channel leakage that persists even after shuffling. Specifically, we construct multiple sets of special malicious ciphertexts, execute decapsulation, and collect side-channel traces during shuffled polynomial reduction. Using a Hamming weight classifier, we build the Hamming weight distribution of the reduced polynomial's coefficients from the side-channel traces. Then, secret key coefficients are inferred by comparing the Hamming weight distributions obtained under different malicious ciphertexts. Our algorithm can recover 2-3 secret key coefficients simultaneously, whereas the bit-flip technique [18], [23] recovers only one message bit at one time.
3) We develop an error-correction strategy that leverages observations on the behavior of the Hamming weight classifier to constrain and shrink the correction search space, and correct Hamming weight distributions with minimal trace repetitions. This allows our secret key recovery algorithm to maintain effectiveness even under less accurate classifiers or in low-SNR environments.

C. Outline

The paper is organized as follows: Section II explains CRYSTALS-Kyber and shuffling protection mechanisms. Section III describes the attack scenarios and threat model. Section IV details the experimental setup, including the hardware and devices used for collecting side-channel information. Section V shows how we found leakage points, segmented traces, and trained a Hamming weight classifier in the profiling stage. Section VI explains secret key recovery and error correction in the attack stage and how to build malicious ciphertexts. Section VII demonstrates the experimental results. Section VIII concludes the paper.

II. Preliminaries

A. CRYSTALS-Kyber

CRYSTALS-Kyber [24] is an IND-CCA2-secure cryptographic algorithm, indicating its resistance to adaptive Chosen Ciphertext Attacks (CCA) through ciphertext indistinguishability. Its security foundation rests on the computational hardness of solving the Module Learning with Errors (Mod-LWE) problem. Kyber consists of two components: KYBER.CPAPKE, a chosen-plaintext attack (CPA)-secure public-key encryption scheme, and KYBER.CCAKEM, a CCA-secure key encapsulation mechanism. Complete algorithmic specifications for KYBER.CPAPKE are presented in Algorithm 1, 2 and 3.

Within the CRYSTALS-Kyber framework, $\mathbb{Z}_q$ denotes the ring of integers modulo prime q, and $R_q = \mathbb{Z}_q[X]/(X^n+1)$ is the quotient ring with $n = 256$, $q = 3329$. The scheme operates on vectors in R_q^k. There are three parameter settings of Kyber: Kyber-512, Kyber-768 and Kyber-1024 corresponding to $k = 2, 3, 4$ respectively, where the rank parameter k is denoted as `KYBER_K`. In this paper, we focus on Kyber-768. Our attack point is the polynomial reduction immediately after computing $v - u \cdot s$. For convenience, we denote the output polynomial of `poly_reduce()` as $\boldsymbol{r}$ in subsequent sections.

Algorithm 1 KYBER.CPAPKE.KeyGen()

Ensure: pk, sk

1: $(\rho, \sigma) \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k\times k}; \rho)$
3: $\mathbf{s}, \mathbf{e} \leftarrow B_{\eta_1}(R_q^{k\times 1}; \sigma)$
4: $t = \text{Encode}_{12}(\mathbf{As} + \mathbf{e})$
5: $s = \text{Encode}_{12}(\mathbf{s})$
6: **return** $(pk = (t, \rho), sk = s)$

Algorithm 2 KYBER.CPAPKE.Enc()

Require: pk, m, r
Ensure: c

1: $\mathbf{t} = \text{Decode}_{12}(t)$
2: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k\times k}; \rho)$
3: $\mathbf{r} \leftarrow B_{\eta_1}(R_q^{k\times 1}; r)$
4: $\mathbf{e_1} \leftarrow B_{\eta_2}(R_q^{k\times 1}; r)$
5: $e_2 \leftarrow B_{\eta_2}(R_q^{1\times 1}; r)$
6: $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e_1}$
7: $v = \mathbf{t}^T\mathbf{r} + e_2 + \text{Decompress}_q(\text{Decode}_1(m), 1)$
8: $c_1 = \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$
9: $c_2 = \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$
10: **return** $c = (c_1, c_2)$

Algorithm 3 KYBER.CPAPKE.Dec()

Require: s, c
Ensure: m

1: $\mathbf{u} = \text{Decompress}_q(\text{Decode}_{d_u}(c_1), d_u)$
2: $v = \text{Decompress}_q(\text{Decode}_{d_v}(c_2), d_v)$
3: $\mathbf{s} = \text{Decode}_{12}(s)$
4: $m = \text{Encode}_1(\text{Compress}_q(v - \mathbf{s}\cdot\mathbf{u}, 1))$
5: **return** m

B. Contermeasure: Shuffling

Shuffling is a widely used randomization-based defense mechanism against side-channel attacks. We adopt the Fisher-Yates algorithm [25] to generate a uniform random permutation sequence by traversing the array in reverse and performing random swaps. The pseudo-random index sequence is employed as the processing sequence for polynomial coefficients, replacing the conventional linear traversal approach. This non-linear execution sequence effectively eliminates temporal correlations between algorithmic physical manifestations (e.g., power consumption, electromagnetic emissions) and data index patterns, thereby substantially enhancing resistance against side-channel cryptanalytic attempts.

We incorporate a function `shuffled_poly_reduce()` to implement shuffling for the `poly_reduce()` operation in the `Kyber.PKE.Decrypt()` execution. The function is illustrated in Listing 1. This implementation first initializes a random permutation sequence via `FY_Gen()`. During the reduction phase, it iterates through this pseudo-randomized index sequence to apply `barrett_reduce()` operations on targeted polynomial coefficients, thereby decoupling temporal execution patterns from the underlying arithmetic logic.

Because shuffling exclusively modifies the coefficient storage and loading sequence in `poly_reduce()`, while preserving the instruction sequence of the core reduction operation `barrett_reduce()` intact, side-channel features generated by index-oblivious operations (e.g., modular multiplication, bit-shifting) exhibit high similarity between the original and the shuffled versions. However, the shuffled version can significantly enhance the complexity of side-channel attacks by disrupting the temporal-data correlation.

```
// Fisher-Yates Shuffle Implementation
void FY_Gen(uint8_t *fylist, int max) {
    for (int i = 0; i < max; i++)
        fylist[i] = i;
    for (int i = max-1; i > 0; i--) {
        int index = rand() % (i+1);
        uint8_t temp = fylist[index];
        fylist[index] = fylist[i];
        fylist[i] = temp;
    }
}
// Barrett Reduction
int16_t barrett_reduce(int16_t a) {
    int32_t t;
    const int32_t v = (1U << 26) / KYBER_Q + 1;
    t = v * a;
    t >>= 26;
    t *= KYBER_Q;
    return a - (int16_t)t;
}
// Polynomial Reduction with Shuffling
void shuffled_poly_reduce(poly *r) {
    uint8_t fylist[KYBER_N];
    FY_Gen(fylist, KYBER_N);
    for (size_t i = 0; i < KYBER_N; i++) {
        size_t i_rand = fylist[i];
        r->coeffs[i_rand] = barrett_reduce(r->
            coeffs[i_rand]);
    }
}
```

Listing 1. Fisher-Yates Shuffle and Polynomial Reduction

III. Threat Model

In this section, we formalize the side-channel attack (SCA)-assisted chosen-ciphertext attack (CCA) scenario in which adversaries aim to exfiltrate the secret key from the target device (Device Under Attack). This is achieved by exploiting side-channel leakage emitted specifically during the decapsulation of CRYSTALS-Kyber when processing chosen ciphertexts. The model assumes the following capabilities and conditions:

1) Adversary can eavesdrop on public communication channels to intercept plaintext communications;
2) Adversary can physically access the target device for side-channel data collection;
3) Adversary can adaptively query the target device to decapsulate chosen ciphertexts;
4) Shuffle countermeasure is disabled on the profiling device during the Profiling Stage;
5) The key pair (pk,sk) remains persistent throughout the Attack Stage, while Fisher-Yates randomized indexes update during each decapsulation operation.

During the Profiling Stage, the adversary intercepts transmitted ciphertexts from public channels and collects side-channel information from the target device during decapsulation `Kyber.KEM.decaps()`. This data is used to train a Hamming weight classifier (we employ a Multi-Layer Perceptron as the classifier in our experiment).

During the Attack Stage, the adversary constructs malicious ciphertexts and queries the target device to decapsulate these ciphertexts using its persistent secret key, capturing side-channel leakage during `poly_reduce()` (Listing 1). The pre-trained classifier then recovers the Hamming weight distribution of the `poly_reduce()` output polynomial $\boldsymbol{r}$, which is a secret-dependent intermediate, and exploits this distribution to reconstruct the secret key.

IV. Experimental Setup

We deploy an STM32F407IG microcontroller (ARM Cortex-M4 32-bit RISC core) operating at 53.76 MHz as the **target device**. Voltage traces are acquired at a sampling rate of 500 MS/s using the Pico 3206D oscilloscope and a voltage probe. All software implementations are compiled using the `arm-none-eabi-gcc` compiler with the optimization flag `-O`.

V. Profiling Stage

The Profiling Stage trains a Hamming weight classifier using side-channel traces from Kyber's `poly_reduce()`. We employ a multilayer perceptron (MLP) as the classifier. The Hamming weight classifier is constructed as follows: (1) acquire side-channel traces from Kyber's `poly_reduce()` at positions $\mathcal{P}_0$ (during NTT) and $\mathcal{P}_1$ (after $v - s \cdot u$ in Line 4 of Algorithm 3), using $\mathcal{P}_0$ traces for MLP pre-training and $\mathcal{P}_1$ traces for fine-tuning; (2) segment each trace into 256 subtraces corresponding to `poly_reduce()` cycles; and (3) train the MLP using these subtraces labeled with the Hamming weight of polynomial $\boldsymbol{r}$'s coefficients (the output of `poly_reduce()`). For trace segmentation, we first establish initial segmentation using leakage points identified by Correlation Power Analysis (CPA) and Welch's t-test, then train the MLP classifier using these segments and perform explainable deep learning techniques to optimize segmentation, enhancing classifier performance.

A. Training Trace Acquisition

To acquire side-channel traces for training our Hamming weight classifier, we generate random plaintext-key pairs and employ the target device detailed in Section IV to perform decapsulation. We collect side-channel traces associated with two invocations of the `poly_reduce()` function within the `Kyber.KEM.decaps()`: (1) We target the `poly_reduce()` within the Number Theoretic Transform (NTT) executed during decryption, denoting this trace collection position as $\mathcal{P}_0$. (2) We target the `poly_reduce()` after computing the critical value $v - u \cdot s$ in Algorithm 3, denoting this trace collection position as $\mathcal{P}_1$. The $\mathcal{P}_0$ traces serve for classifier pre-training, while the $\mathcal{P}_1$ traces are used for classifier fine-tuning and testing.

In our threat model, we assume that the shuffle countermeasure is disabled on the profiling device during the profiling stage. An adversary intercepts ciphertexts via public channel eavesdropping. Crucially, the first NTT in `Decrypt()` operates without the secret key, making its all intermediate variables computable from ciphertexts; thus, $\mathcal{P}_0$ `poly_reduce()` output is derivable. In contrast, $\mathcal{P}_1$ `poly_reduce()` output remains unobservable due to secret-key-dependent precomputation. We can train an MLP model on the $\mathcal{P}_0$ `poly_reduce()` output derived from intercepted ciphertexts and $\mathcal{P}_0$ side-channel traces. Training data scales arbitrarily with side-channel traces and ciphertext captures. Critically, by collecting $\mathcal{P}_0$ training data directly from the target device, we eliminate cross-device bias, ensuring model accuracy is not compromised by hardware variations. Only minimal $\mathcal{P}_1$ traces from the target device are then needed to fine-tune the model for deployment at $\mathcal{P}_1$.

We generated 1000 random key-ciphertext pairs, capturing two traces $T_j^{\mathcal{P}_0}$ and $T_j^{\mathcal{P}_1}$ at two positions per decapsulation for $0 \leq j < 1000$. Each trace was partitioned into 256 subtraces $\{T_j[i]\}_{i=0}^{255}$, corresponding to the 256 cycles in `poly_reduce()`. For clarity, we define $T_j[i]$ as the i-th subtrace ($0 \leq i < 256$) of j-th trace T_j and $T_j[t]$ as the value of T_j at the t-th sampling point ($0 \leq t < \text{length}(T_j)$).

The input of the model is subtrace $T_j[i]$ while the label is the Hamming weight of corresponding $r_j[i]$, where $\boldsymbol{r}$ represents `poly_reduce()` output polynomial and $r_j[i]$ represents the i-th coefficient of the j-th polynomial. We obtained 256000 subtraces from $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively. The dataset from $\mathcal{P}_0$ traces was partitioned into an 80% training subset $\mathcal{T}_{\text{train}}$ (204800 subtraces) and a 20% test subset $\mathcal{T}_{\text{test}}^{\mathcal{P}_0}$ (51200 subtraces). The dataset from $\mathcal{P}_1$ traces was partitioned into an 20% fine-tuning subset $\mathcal{T}_{\text{fine-tune}}$ (51200 subtraces) and a 80% test subset $\mathcal{T}_{\text{test}}^{\mathcal{P}_1}$ (204800 subtraces).

B. Location of Points of Interest and Segmentation

The `poly_reduce()` function involves over 256 cycles. Precise localization of side-channel leakage points and parti-

tioning of corresponding subtraces $T_j[i]$ are critical for feature extraction quality, directly impacting model training performance and classification accuracy. To achieve this, we first establish initial segmentation using leakage points identified by Correlation Power Analysis (CPA) [26] and Welch's t-test [27], then perform explainable deep learning techniques to optimize segmentation.

1) Correlation Power Analysis: The Pearson correlation coefficient (PCC) measures the strength of the linear relationship between two continuous variables. To identify the leakage point associated with the i-th coefficient of $\boldsymbol{r}$, we compute the PCC between the side-channel traces in the training dataset $\mathcal{T}_{\text{train}}$ and the Hamming weights of corresponding $r_j[i]$ for each sampling point t ($0 \leq t < \text{length}(T_j)$):

$$\text{Corr}_i[t] = \frac{\sum_{j=0}^{999}(T_j[t] - \bar{T[t]})(\text{HW}(r_j[i]) - \bar{\text{HW}})}{\sqrt{\sum_{j=0}^{999}(T_j[t] - \bar{T[t]})^2 \sum_{j=0}^{999}(\text{HW}(r_j[i]) - \text{HW}(\bar{r[i]}))^2}} \quad (1)$$

where $\bar{T[t]}$ and $\bar{HW(r[i])}$ denote the means of $\mathcal{T}_{\text{train}}[t]$ and $\{HW(r_j[i])\}_{j=0}^{999}$ respectively. The sampling point with the maximal $|\text{Corr}_i[t]|$ value is identified as the leakage point.

2) Welch's t-test: Welch's t-test assesses the significance of mean differences between two independent samples with unequal variances. To identify the leakage point associated with the i-th coefficient of $\boldsymbol{r}$, the training dataset $\mathcal{T}_{train}$ is partitioned into two sets $\mathcal{T}_0$ and $\mathcal{T}_1$.

$$\mathcal{T}_0 = \{T_j | T_j \in \mathcal{T}_{train}, \text{HW}(r_j[i]) < 6\}, \quad (2)$$

$$\mathcal{T}_1 = \{T_j | T_j \in \mathcal{T}_{train}, \text{HW}(r_j[i]) > 6\} \quad (3)$$

Then, Welch's t-statistic for each sampling point t is computed as:

$$\text{TStat}_i[t] = \frac{\bar{T_0[t]} - \bar{T_1[t]}}{\sqrt{\frac{\sigma_0^2[t]}{n_0} + \frac{\sigma_1^2[t]}{n_1}}}, 0 \leq i < 256 \quad (4)$$

where $\bar{T_0[t]}$ and $\bar{T_1[t]}$ represent the means of $\mathcal{T}_0[t]$ and $\mathcal{T}_1[t]$, and n_0 and n_1 represent the sizes of $\mathcal{T}_0[t]$ and $\mathcal{T}_1[t]$ respectively. The sampling point with the maximal $|\text{TStat}_i[t]|$ value, rejecting the null hypothesis of equal means, is identified as the leakage point.

3) Trace Segmentation: Figure 1(a) shows the average of all raw traces in the training set $\mathcal{T}_{\text{train}}$. It reveals a distinct periodic structure corresponding to the 256 cycles in `poly_reduce()`, with each period spanning approximately 234 sampling points. The detailed waveform is shown in Figure 2(a).

Concurrently, Figure 1(b) displays the Pearson Correlation Coefficient ($\text{Corr}_i[t]$) for specific indices $i \in \{8, 40, 72, 104, 136, 168, 200, 232\}$, where peaks identify $r[i]$ leakage points. Similarly, Figure 1(c) shows Welch's t-statistic ($\text{TStat}_i[t]$) for the same indices, where its peaks also identify $r[i]$ leakage points. Critically, both CPA and Welch's t-test waveforms exhibit nearly identical shapes and peak positions. Moreover, the cycle period of 234 observed in $\bar{T}_{\text{train}}$ exactly matches the peak intervals in Welch's t-statistic. This precise synchronization and temporal alignment is visually confirmed in Figure 2, which proves that we accurately capture the cryptographic operation's true cycle structure. This consistency establishes a robust foundation for subsequent trace segmentation.

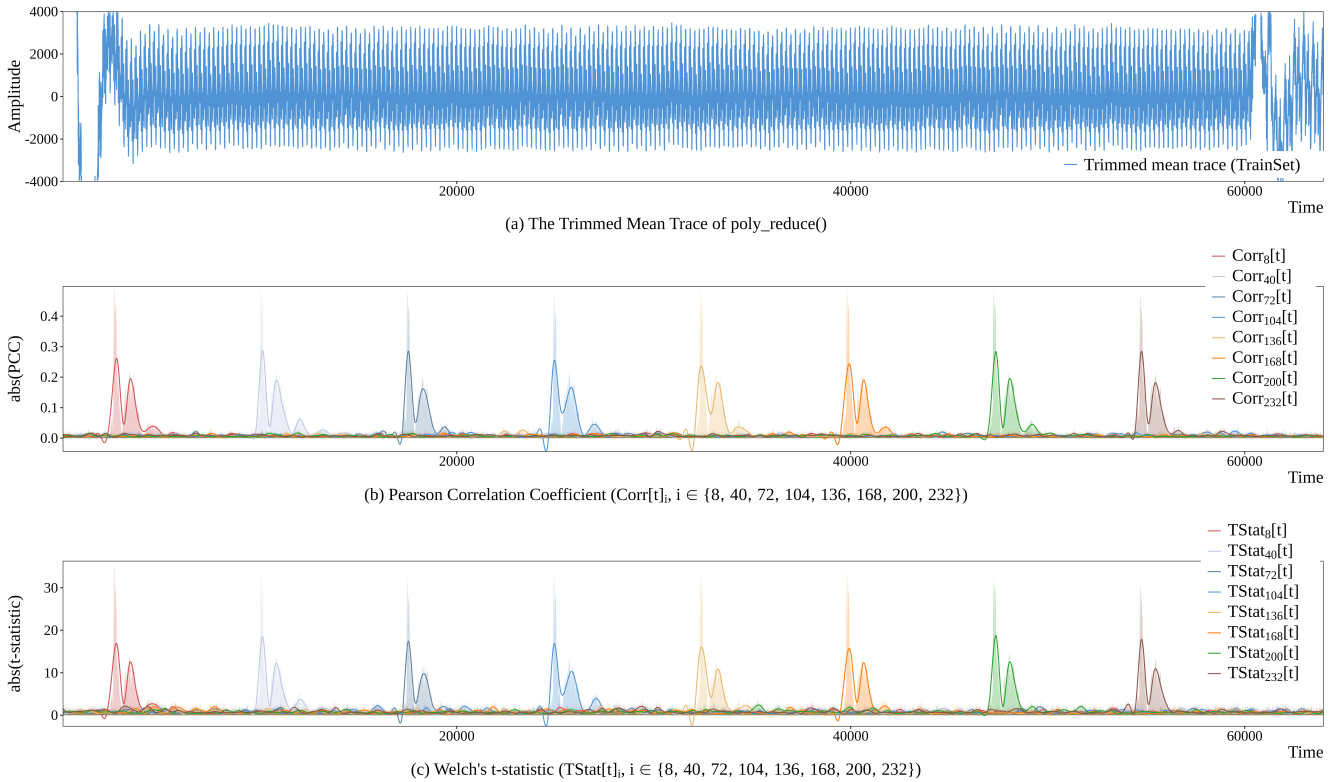
Based on analysis, our initial trace segmentation scheme employs the rising edges of Welch's t-statistic as segmentation starting points. Each segment spans a fixed length of 234 sampling points, corresponding to one complete cycle in `poly_reduce()`. These segments subsequently serve as input vectors for our MLP classifier, with the target labels being the Hamming weights of $\boldsymbol{r}$ as detailed in Section V-A. The segmentation scheme is visualized by the gray demarcation line in Figure 2.

To exploit the detected leakage, we constructed an MLP classifier with the architecture specified in Table I. When evaluated on the testset $\mathcal{T}_{\text{test}}^{\mathcal{P}_0}$, the model achieved a prediction accuracy of 96.23%. To probe the model's decision mechanism and identify important features, we employed two diagnostic methods: First, for correctly classified outputs, we computed the gradient of the model output with respect to the input vector via backpropagation (visualized in Figure 3(a) Grey Line). This gradient effectively quantifies the sensitivity of the output prediction to infinitesimal perturbations at each sampling point position, thus serving as a direct measure of feature importance. Second, we extracted the weights associated with the input layer of the trained MLP model (visualized in Figure 3(b) Grey Line), as their magnitude distribution across the input dimension implicitly reflects the relative importance assigned by the model to each input feature during inference, [17], [28] similarly employed this technique for Points of Interest (POI) localization.

This diagnostic evidence identifies a feature distribution shift within the original segmentation scheme, where critical leakage information is predominantly concentrated in the left-most region of the input vector. This indicates that side-channel leakage was originally clustered near the left boundary, potentially omitting some leakage points beyond this boundary. So, we shifted the segmentation starting points leftward by 105 sampling points while maintaining a subtrace length of 234 sampling points. This adjustment centers the critical leakage information within the subtrace and recovers previously omitted leakage points beyond the original left boundary. The modified segmentation scheme is visualized by the orange demarcation line in Figure 3. This optimized segmentation improved classifier accuracy by 0.88% to 97.11%, attributable to the inclusion of previously omitted side-channel leakage points. The 0.88% improvement in Hamming weight classifier accuracy holds critical engineering value, because this gain reduces the key enumeration space and thereby increases attack efficiency in specific attack scenarios.

C. Network Architecture and Training

The architecture of multilayer perceptron (MLP) for the Hamming weight classification is detailed in Table I. The model was trained using a batch size of 128 for up to 200 epochs, incorporating an early stopping strategy with a patience of 10 to prevent overfitting. We used the Adam



(a) The Trimmed Mean Trace of poly_reduce()

(b) Pearson Correlation Coefficient (Corr[t]$_i$, i ∈ {8, 40, 72, 104, 136, 168, 200, 232})

(c) Welch's t-statistic (TStat[t]$_i$, i ∈ {8, 40, 72, 104, 136, 168, 200, 232})

Fig. 1. The Trimmed Mean Trace of `poly_reduce()`, Pearson Correlation Coefficient and Welch's t-statistic. For clarity and visual conciseness, only a representative subset of PCC and Welch's t-statistic is displayed.

TABLE I
MLP NETWORK ARCHITECTURE

Layer Type	Input Shape	Output Shape	# Parameters
Dense 1	234	1024	240,640
Batch Normalization 1	1024	1024	4,096
ReLU 1	1024	1024	0
Dense 2	1024	512	524,800
Batch Normalization 2	512	512	2,048
ReLU 2	512	512	0
Dense 3	512	256	131,328
Batch Normalization 3	256	256	1,024
ReLU 3	256	256	0
Dense 4	256	128	32,896
Batch Normalization 4	128	128	512
ReLU 4	128	128	0
Dense 5	128	12	1,548
Total Parameters			938,892
Trainable Parameters			935,052

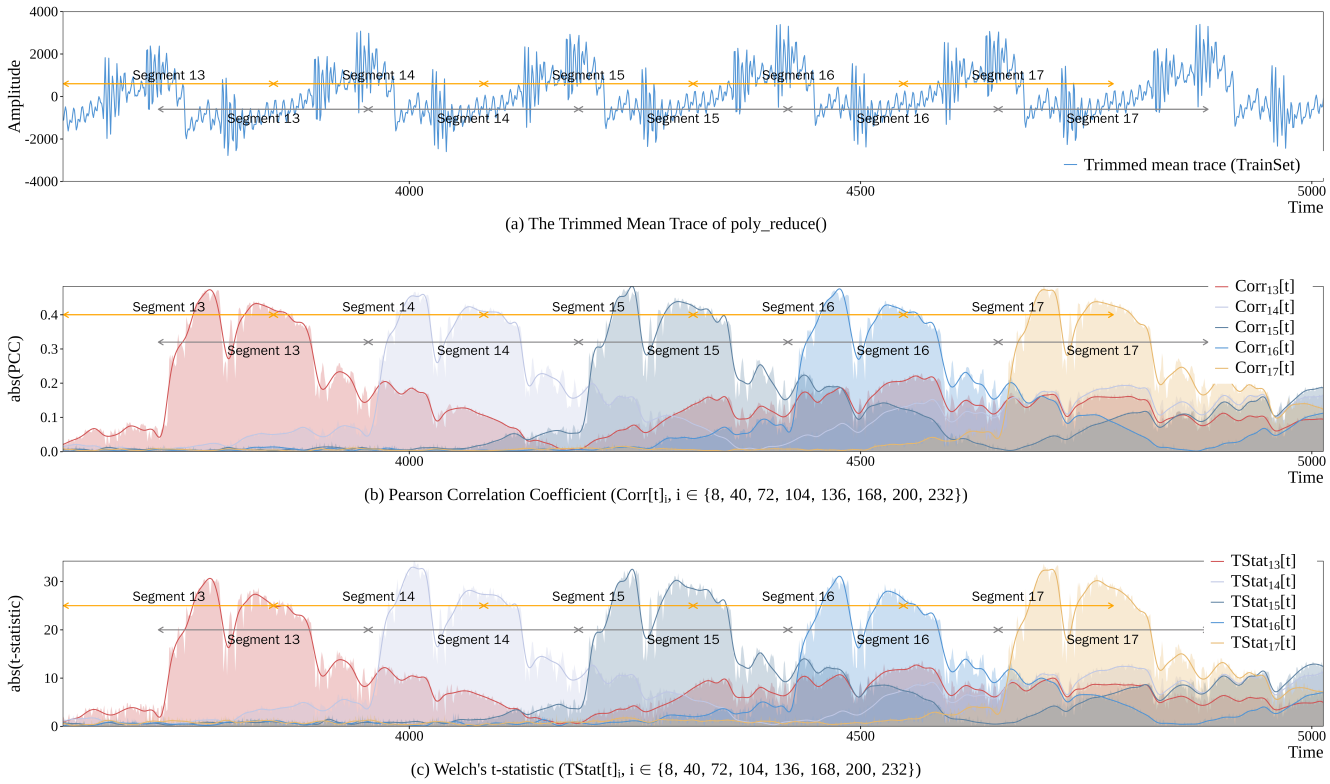
optimizer with an initial learning rate of 0.001 and a learning rate scheduler that reduced the rate by a factor of 0.1 after 3 epochs without validation accuracy improvement, enforcing a minimum learning rate of 10^{-10}. Cross-entropy loss was employed as the optimization objective. The final model was selected based on peak validation accuracy observed during training.

VI. ATTACK STAGE

A. Secret Key Recovery Algorithm

Although shuffling-based countermeasures alter side-channel information by randomizing processing order, they retain higher-order statistical characteristics. Attackers can thus extract secrets by analyzing inter-trace statistical differential. In this section, we elucidate step by step the methodology by which we leverage this vulnerability to recover the secret key and the method for constructing special malicious ciphertexts. From Step 1 to Step 6, we construct two sets of specially constructed malicious ciphertexts and supply them to the target device for decapsulation, and collect side-channel traces during the execution of `poly_reduce()`. Using the Hamming weight classifier trained in Profiling Stage, we infer the Hamming weight distribution of the `poly_reduce()` output polynomial $\boldsymbol{r}$ from the collected traces. By comparing the Hamming weight distributions obtained under two malicious ciphertexts, we can recover two or three secret key coefficients simultaneously. This process is repeated until all secret key coefficients are determined. However, when the classifier's accuracy is not optimal, this attack becomes highly inefficient. To address this, we developed an error correction method that reconstructs accurate Hamming weight distributions with minimal trace capture repetitions, as elaborated in Step 2* and Step 5*.

Step 1: Build basic malicious ciphertext: To recover the p-th polynomial $\boldsymbol{s} = \sum_{i=0}^{255} s[i] \cdot x^i \triangleq \{s[0], s[1], \cdots, s[255]\}_{256}$

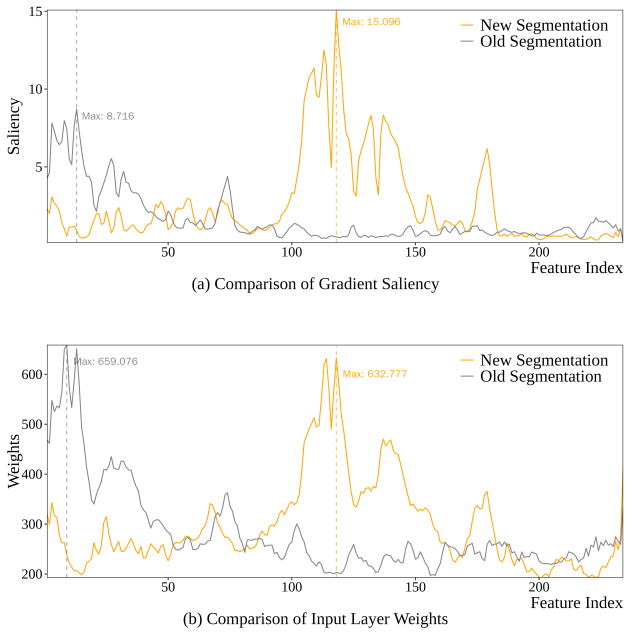


(a) The Trimmed Mean Trace of poly_reduce()

(b) Pearson Correlation Coefficient (Corr[t]$_i$, i ∈ {8, 40, 72, 104, 136, 168, 200, 232})

(c) Welch's t-statistic (TStat[t]$_i$, i ∈ {8, 40, 72, 104, 136, 168, 200, 232})

Fig. 2. Partial Trimmed Mean Trace of `poly_reduce()`, Pearson Correlation Coefficient and Welch's t-statistic with Two Segmentations. Orange demarcation lines denote new segmentation boundaries, while gray lines represent old segmentation boundaries.



(a) Comparison of Gradient Saliency

(b) Comparison of Input Layer Weights

Fig. 3. Comparison of Feature Importance Analysis

of the secret key, where $0 \leq p < \texttt{KYBER\_K}$ and $s[i] \in \{-2, -1, 0, 1, 2\}$, we construct a basic malicious ciphertext $ct = (\mathbf{u}, \boldsymbol{v})$. Here, $\mathbf{u} \in \mathbb{R}_q^{3\times 1}$ and $\boldsymbol{v} \in \mathbb{R}_q$ are defined as:

$$\mathbf{u} = \begin{cases} (\boldsymbol{ku}, \mathbf{0}, \mathbf{0}) \in \mathbb{R}_q^{3\times 1} \text{ for } p = 0 \\ (\mathbf{0}, \boldsymbol{ku}, \mathbf{0}) \in \mathbb{R}_q^{3\times 1} \text{ for } p = 1 \\ (\mathbf{0}, \mathbf{0}, \boldsymbol{ku}) \in \mathbb{R}_q^{3\times 1} \text{ for } p = 2 \end{cases} \tag{5}$$

and

$$\boldsymbol{v} = \sum_{i=0}^{255} kv_0 \cdot x^i \triangleq \{kv_0, kv_0, \cdots, kv_0\}_{256} \tag{6}$$

where, $\boldsymbol{ku}$ and $\mathbf{0}$ denote the monomial $ku + \sum_{i=1}^{255} 0 \cdot x^i \triangleq \{ku, 0, \cdots, 0\}_{256}$ and the zero polynomial $\sum_{i=0}^{255} 0 \cdot x^i \triangleq \{0, 0, \cdots, 0\}_{256}$, respectively

Step 2: Acquisition and analysis of side-channel information (basic malicious ciphertext): The target device described in Section IV performs decapsulation of the basic malicious ciphertext. Specifically, Kyber first unpacks the byte-packed ciphertext ct into polynomials $\mathbf{u}$ and $\boldsymbol{v}$, then invokes `Kyber.PKE.Decrypt()` to compute

$$\boldsymbol{r} = \boldsymbol{v} - \text{NTT}^{-1}(\text{NTT}(\mathbf{u}) \circ \text{NTT}(\mathbf{s})) \tag{7}$$

$$= \{kv_0 - ku \cdot s[0], \cdots, kv_0 - ku \cdot s[255]\}_{256} \tag{8}$$

We collect the side-channel trace T when `Kyber.KEM.Decaps()` invokes `poly_reduce()` (at Position $\mathcal{P}_1$), and then segment T into 256 subtraces $T[i]$ $(0 \leq i < 256)$ using the method described in Section V-B.

Each subtrace $T[i]$ was fed into the Hamming weight classifier (the MLP model trained in Profiling Stage) to

obtain the Hamming weight $\mathrm{HW}(r[i])$ corresponding to each coefficient $r[i]$. The Hamming weight frequency distribution vector $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ is then statistically analyzed; each entry $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))[w]$ corresponds to the count of $\boldsymbol{r}$'s coefficients exhibiting a specific Hamming weight value w for $w \in \{0, 1, \ldots, 11\}$. Since $\boldsymbol{r}$ is uniquely determined by the secret key polynomial $\boldsymbol{s}$ and ciphertext ct, this distribution may also be denoted as $\boldsymbol{Dist}(\boldsymbol{s}, ct)$.

$$\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r})) = \begin{pmatrix} \#\{i \mid \mathrm{HW}(r[i]) = 0\} \\ \#\{i \mid \mathrm{HW}(r[i]) = 1\} \\ \vdots \\ \#\{i \mid \mathrm{HW}(r[i]) = 11\} \end{pmatrix} \tag{9}$$

Step 2*: Correct erroneous Hamming weight distribution (basic malicious ciphertext, Optional): When the Hamming weight classifier's accuracy is not optimal, Step 2 may yield an incorrect $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$. Since the permutation rule of the shuffle operation is updated with each execution, the error distribution in each measurement is random and independent. Even if Step 2 is repeated multiple times, it remains challenging to obtain a completely accurate $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ in any single run. However, two characteristics of the MLP classifier can be leveraged to rapidly correct $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$: (1) the classifier may misclassify $T[i]$, whose true label is $\mathrm{HW}(r[i])$, as $\mathrm{HW}(r[i]) \pm 1$; (2) the top-2 cumulative accuracy of the classifier approaches 100%. These properties are analyzed in detail in Section VII-A.

For Kyber-768 with $s[i] \in \{-2, -1, 0, 1, 2\}$, $r[i] = kv_0 - ku \cdot s[i]$ takes one of five possible values. Using Row 1 parameters from Table III ($ku = 1586$, $kv[0] = 1040$), $\mathrm{HW}(r[i]) \in \{2, 4, 6, 7, 9\}$ (as listed in Table II). Let us first consider several basic scenarios:

1) If the classifier predicts an invalid Hamming weight $\mathrm{HW} = 10$, the only valid adjacent weight is 9. Given that the classifier tends to misclassify toward neighboring values, the prediction is corrected to $\mathrm{HW} = 9$.
2) If the classifier predicts an invalid Hamming weight $\mathrm{HW} = 3$, the true prediction must be either $\mathrm{HW} = 2$ or $\mathrm{HW} = 4$ (both valid). Since the top-2 cumulative accuracy approaches 100%, if $\mathrm{HW} = 3$ is the top prediction and $\mathrm{HW} = 2$ is second, the prediction is corrected to $\mathrm{HW} = 2$.
3) When the classifier predicts $\mathrm{HW} = 6$ (corresponding to $s[i] = -2$) and $\mathrm{HW} = 7$ (corresponding to $s[i] = 2$) is the second prediction, ambiguity arises: the classifier may be correct with $\mathrm{HW} = 6$, or it may be mistaken and the true value is $\mathrm{HW} = 7$. This case cannot be resolved using the basic malicious ciphertext constructed from Row 1 parameters. We therefore switch to Row 3 parameters from Table III ($ku = 1297$, $kv[0] = 1248$), under which $s[i] = -2$ corresponds to $\mathrm{HW}(r[i]) = 10$ and $s[i] = 2$ corresponds to $\mathrm{HW}(r[i]) = 2$. These extreme Hamming weights (10 vs. 2) exhibit minimal classifier confusion. By strategically alternating between parameter sets, $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ can be reconstructed efficiently with minimal repetitions. We refer to the basic malicious ciphertext constructed from Row 3 parameters as the auxiliary basic malicious ciphertext.

TABLE II
$\mathrm{HW}(r[i])$ FOR DIFFERENT $s[i]$ VALUES UNDER OPTIMAL PARAMETERS

ku	kv_0	$s[i]$				
		-2	-1	0	1	2
1586	1040	6	9	2	4	7
1297	1248	10	5	4	7	2

Building upon the analysis above, we formalize the error correction procedure for the Hamming weight distribution $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ corresponding to the basic malicious ciphertext.

A single classification error in $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ is characterized as an atomic error. Specifically, when the Hamming weight classifier misclassifies a subtrace $T[i]$ whose true Hamming weight is x as y (where $y \in \{x-1, x+1\}$), we denote this event as $e_{x \to y}$. This atomic error is represented by a 1×12 vector containing -1 at position x, $+1$ at position y, and zeros elsewhere.

Given that the top-2 cumulative accuracy of the classifier approaches 100%, if the top prediction is erroneous, the second prediction is highly likely to be correct. We therefore scan the classifier's predictions across all 256 subtraces, treating the second prediction as the true value x and the top prediction as the misclassified value y, thereby identifying atomic errors. The frequency of each atomic error type is recorded. Note that certain atomic errors are invalid and are excluded from consideration. Specifically, we exclude errors where the true value x falls outside the possible value range of $\mathrm{HW}(r[i])$, as well as errors where the misclassified value y is not adjacent to the true value x ($y \notin \{x-1, x+1\}$). The error correction space $\mathcal{C}$ is defined as the set of all possible finite linear combinations of valid atomic errors, representing the complete set of potential classification inaccuracies in a single measurement.

We perform five independent repetitions of side-channel acquisition and analysis (as described in Step 2) for both the basic malicious ciphertext and the auxiliary basic malicious ciphertext, resulting in ten potentially erroneous instances of $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$. For each repetition, we analyze its error correction space $\mathcal{C}$ and exhaustively evaluate each possible error combination within $\mathcal{C}$. Each error combination is subtracted from the measured $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ to obtain a corrected distribution. A corrected distribution is admitted to the candidate set only if it contains non-zero values exclusively at the five valid Hamming weights corresponding to the ciphertext parameters used (i.e., $\{2, 4, 6, 7, 9\}$ for Row 1 parameters). Finally, the $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ that appears most frequently in the candidate set across all repetitions is selected as the correct distribution.

Step 3: Build derived malicious ciphertext: To simultaneously recover random k coefficients $(s[i_1], \cdots, s[i_k])$ of the p-th polynomial of the secret key, where the k target positions $(i_1, \cdots, i_k)$ are pairwise distinct and can be randomly selected , we construct a derived malicious ciphertext $ct' = (\mathbf{u}, \boldsymbol{v}')$ by

modifying k coefficients of $\boldsymbol{v}$. Specifically, for $\boldsymbol{v}'$:

$$\boldsymbol{v}' = \sum_{i=0}^{255} v'[i] \cdot x^i, v'[i] = \begin{cases} kv_\ell & \text{if } i = i_\ell, \ell \in \{1, \ldots, k\} \\ kv_0 & \text{otherwise} \end{cases} \quad (10)$$

This manipulation induces an exploitable differential in the coefficients of the output polynomial $\boldsymbol{r}'$ from `poly_reduce()`.

Step 4: Acquisition and analysis of side-channel information (derived malicious ciphertext): Repeat the procedure from Step 2. $\boldsymbol{r}'$ shows differences in k coefficients compared to $\boldsymbol{r}$ which are caused by modifications in the derived malicious ciphertext:

$$\begin{cases} r[i_1] = kv_0 - ku \cdot s[i_1] \rightarrow r'[i_1] = kv_1 - ku \cdot s[i_1] \\ r[i_2] = kv_0 - ku \cdot s[i_2] \rightarrow r'[i_2] = kv_2 - ku \cdot s[i_2] \\ \quad \vdots \\ r[i_k] = kv_0 - ku \cdot s[i_k] \rightarrow r'[i_k] = kv_k - ku \cdot s[i_k] \end{cases} \quad (11)$$

Compute the frequency distribution $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}'))$, alternatively denoted as $\boldsymbol{Dist}(\boldsymbol{s}, ct')$.

Step 5: Compute the Hamming weight distribution differential and query the mapping table to recover the k coefficients of the secret key: Compute the Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s}) = \boldsymbol{Dist}(\boldsymbol{s}, ct') - \boldsymbol{Dist}(\boldsymbol{s}, ct)$. Specifically, $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ is a vector where each entry $\Delta\boldsymbol{Dist}(\boldsymbol{s})[w]$ (for $w \in \{0, 1, \ldots, 11\}$) represents the difference in counts of Hamming weight w between the vectors $\boldsymbol{r}$ and $\boldsymbol{r}'$ across all indices $i \in [0, 256)$. That is:

$$\Delta\boldsymbol{Dist}(\boldsymbol{s}) = \begin{pmatrix} \#\{i|\mathrm{HW}(r'[i])=0\} - \#\{i|\mathrm{HW}(r[i])=0\} \\ \#\{i|\mathrm{HW}(r'[i])=1\} - \#\{i|\mathrm{HW}(r[i])=1\} \\ \vdots \\ \#\{i|\mathrm{HW}(r'[i])=11\} - \#\{i|\mathrm{HW}(r[i])=11\} \end{pmatrix} \quad (12)$$

We use $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ as the lookup key and the secret key coefficient tuple $(s[i_1], \ldots, s[i_k])$ as the value. By querying the pre-generated mapping table (see Table V or VI first two columns), which maps the Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ to the k coefficients of the secret key $(s[i_1], \ldots, s[i_k])$, we recover the k secret key coefficients.

Step 5*: Correct Erroneous Hamming Weight Distribution differential (derived malicious ciphertext, Optional): Similar to Step 2*, classification errors during the analysis of the derived malicious ciphertext can induce an erroneous Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$. Correcting $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ is equivalent to correcting $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}'))$ corresponding to the derived malicious ciphertext, since $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ has already been accurately determined and corrected in Steps 2 and 2*. However, a key distinction exists: while Step 2* operated without a reliable reference distribution, we now exploit the structural constraint that $\boldsymbol{r}'$ differs from $\boldsymbol{r}$ in only k coefficients. This sparsity of differences imposes strong constraints on the possible values of $\Delta\boldsymbol{Dist}(\boldsymbol{s})$, which can be leveraged to enhance the error correction process.

We maintain the definitions of atomic errors and the error correction space $\mathcal{C}$ established in Step 2*. The error correction procedure proceeds as follows. We perform at most M independent repetitions of side-channel acquisition and analysis for the derived malicious ciphertext (as described in Step 4), resulting in M potentially erroneous instances of $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}'))$. After each repetition m ($1 \leq m \leq M$), we iterate over all possible secret key coefficient tuples $\boldsymbol{s} = (s[i_1], \ldots, s[i_k]) \in \{-2, -1, 0, 1, 2\}^k$. For each hypothesis $\boldsymbol{s}$, we determine the corresponding valid Hamming weight set for $\mathrm{HW}(r'[i])$ based on the derived malicious ciphertext, which will be used to filter valid atomic errors. We then fetch atomic errors from the classifier's predictions and construct the error correction space $\mathcal{C}_{m,\boldsymbol{s}}$ by forming linear combinations of valid atomic errors.

Subsequently, we exhaustively evaluate each possible error combination $e \in \mathcal{C}_{m,\boldsymbol{s}}$. Each error vector e is subtracted from the measured $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}'))_m$ to obtain a corrected distribution: $\boldsymbol{Dist}_{\text{corr}}(\mathrm{HW}(\boldsymbol{r}')) = \boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}'))_m - e$. Using $\boldsymbol{Dist}_{\text{corr}}(\mathrm{HW}(\boldsymbol{r}'))$ and the previously corrected $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$, we compute the corrected differential:

$$\Delta\boldsymbol{Dist}_{\text{corr}} = \boldsymbol{Dist}_{\text{corr}}(\mathrm{HW}(\boldsymbol{r}')) - \boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r})). \quad (13)$$

This corrected differential is then compared to the expected $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ obtained from the precomputed mapping table for hypothesis $\boldsymbol{s}$. If $\Delta\boldsymbol{Dist}_{\text{corr}}$ matches $\Delta\boldsymbol{Dist}(\boldsymbol{s})$, $\Delta\boldsymbol{Dist}_{\text{corr}}$ is added to the candidate set $\mathcal{C}_m$ for repetition m.

After processing all hypotheses and error combinations for repetition m, we obtain a candidate set $\mathcal{C}_m$. The overall candidate set across all previous repetitions is the union $\mathcal{C} = \bigcup_{i=1}^{m} \mathcal{C}_i$. The correction process iterates over repetitions until the candidate set $\mathcal{C}$ contains only one unique candidate, which is then selected as the correct secret key coefficient tuple and the corresponding $\Delta\boldsymbol{Dist}(\boldsymbol{s})$. This iterative pruning leverages the collective constraints from multiple independent measurements to eliminate spurious candidates that arise from random classification errors in individual traces.

However, empirical analysis in Section VII-B indicates that a single malicious ciphertext is often inadequate for achieving robust error correction across the entire key space of tuples $\boldsymbol{s} \in \{-2, -1, 0, 1, 2\}^k$. This limitation stems from the intricate entanglement of error spaces across different secret key coefficient tuples, as we discussed in Section VI-B. To overcome this, we introduce a cascaded correction strategy that employs two malicious ciphertexts with complementary corrective capabilities.

The two ciphertexts are carefully selected to ensure their error profiles are orthogonal, meaning that each excels at correcting distinct subsets of the key space. The correction process begins by applying the above procedure to the first malicious ciphertext over at most M independent repetitions, yielding a candidate set $\mathcal{C}^{(1)}$. If $\mathcal{C}^{(1)}$ converges to a unique candidate, the process terminates successfully. If multiple candidates remain, the secondary malicious ciphertext is deployed and yield a candidate set $\mathcal{C}^{(2)}$. The final key tuple is derived from their intersection $\mathcal{C} = \mathcal{C}^{(1)} \cap \mathcal{C}^{(2)}$.

This sequential approach ensures that the first ciphertext performs the bulk of the initial pruning, while the secondary ciphertext acts as a specialized filter to resolve residual uncertainties. As empirically validated in Section VII-B, this approach proves highly effective.

Collectively, the techniques presented in Step 2* and Step 5* ensure robust error correction while minimizing the number of required side-channel acquisitions. By systematically leveraging the classifier's error patterns, our method efficiently reconstructs the true Hamming weight distributions critical for successful key recovery.

Step 6: Recover the complete secret key: Repeat Steps 3 to 5*, each time selecting k new distinct indices as targets and modifying the k coefficients of the derived malicious ciphertext to recover the corresponding k coefficients of the secret key. Iterate until the full secret key is recovered. The theoretical minimum number of attacks is $\lfloor \text{KYBER\_N}/k \rfloor * \text{KYBER\_K}$, while the practical attack efficiency depends on parameter selection, the device's side-channel exploitability, and classifier performance.

B. Malicious Ciphertext Choice and Mapping Table Construction

Malicious ciphertexts in Steps 1 and 3 induce an exploitable Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$. The mapping table, generated according to malicious ciphertext parameters ku and kv, is applied in Step 5 to correlate the observed differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ with the secret key coefficient tuple $(s[i_1], \ldots, s[i_k])$. Consequently, careful selection of malicious ciphertext parameters ku and kv is required. This section will specify the mapping table construction method and the parameter selection constraints.

The mapping table is generated by exhaustively enumerating every $\boldsymbol{s} = (s[i_1], \ldots, s[i_k]) \in \{-2, -1, 0, 1, 2\}^k$ and computing its corresponding unique $\Delta\boldsymbol{Dist}(\boldsymbol{s})$. For each $\boldsymbol{s}$, we first initialize $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ as a zero vector spanning all possible Hamming weights, then compute $r[i_j] = kv_0 - ku \cdot s[i_j]$ and $r'[i_j] = kv_j - ku \cdot s[i_j]$ for $j \in \{1, \ldots, k\}$ and derive their Hamming weights $\text{HW}(r[i_j])$ and $\text{HW}(r'[i_j])$. After that, we modify $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ by decrementing by 1 at $\text{HW}(r[i_j])$ while simultaneously incrementing by 1 at $\text{HW}(r'[i_j])$, which means the modification in the derived malicious ciphertext leads to the substitution of $\text{HW}(r[i_j])$ with $\text{HW}(r'[i_j])$ within the Hamming weight distribution, thereby cumulatively constructing a Hamming weight distribution differential. The pre-computed mapping table for malicious ciphertext parameters in Row 1 of Table III ($ku = 1586$, $kv = [1040, 1873, 2497]$) is shown in the first two columns of Table V.

The core constraint in constructing the mapping table requires establishing a strict bijective mapping between Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ and secret key coefficients $(s[i_1], \ldots, s[i_k])$. Formally, for any two distinct coefficients combinations $\boldsymbol{s_1}, \boldsymbol{s_2} \in \{-2, -1, 0, 1, 2\}^k$:

$$\forall \boldsymbol{s_1} \neq \boldsymbol{s_2} \in \{-2, -1, 0, 1, 2\}^k, \Delta\boldsymbol{Dist}(\boldsymbol{s_1}) \neq \Delta\boldsymbol{Dist}(\boldsymbol{s_2}) \quad (14)$$

This bijection ensures a one-to-one correspondence between valid Hamming weight distribution differential and unique secret key coefficient tuple, eliminating recovery ambiguity.

Furthermore, an additional constraint in constructing the mapping table lies in enhancing its fault tolerance, aiming to minimize the impact of classifier errors (as described in Steps 2* and 5*) on secret key recovery. The core idea is to ensure that the error spaces of different candidate secret key coefficient tuples are as mutually disjoint as possible. Here, the error space for a specific candidate $\boldsymbol{s} = (s[i_1], \ldots, s[i_k])$ is defined as the set of corresponding $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ plus all integer linear combinations of possible atomic errors, denoted as $\mathcal{E}(\boldsymbol{s})$.

TABLE III
OPTIMAL ATTACK PARAMETERS FOR KYBER768 ROUND 2

Attack Count (k)	ku	kv
2	1586	[1040 1873 2497]
2	1743	[1040 1873 2497]
2	1297	[1248, 0, 2497]
3	1489	[416 0 1665 1873]
3	1840	[416 0 1665 1873]

To quantify fault tolerance performance, we introduce the error space overlap metric, defined as the number of unordered pairs $(\boldsymbol{s_i}, \boldsymbol{s_j})$ with $\boldsymbol{s_i} \neq \boldsymbol{s_j}$ whose error spaces exhibit significant overlap. This occurs when multiple errors $e \in \mathcal{E}(\boldsymbol{s_i})$ satisfy $\Delta\boldsymbol{Dist}(\boldsymbol{s_i}) + e = \Delta\boldsymbol{Dist}(\boldsymbol{s_j})$, indicating that under specific misclassification patterns, the ideal differential of $\boldsymbol{s_i}$ corrupted by errors could be mistaken for the ideal differential of $\boldsymbol{s_j}$, or vice versa. Therefore, the optimization objective is to select parameters (ku, kv) such that the resulting mapping table not only satisfies the fundamental bijection constraint but also minimizes the number of intersecting error space pairs.

Through systematic enumeration of candidate malicious ciphertext parameters ku and kv combined with computation and comparison of their error space overlap metrics, the parameters listed in Table III were selected as optimal. They simultaneously satisfy two criteria: (1) maintaining a strict bijection between $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ and $\boldsymbol{s}$, and (2) demonstrating near-disjoint error spaces, for nearly all $\boldsymbol{s}$, $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ lies strictly within its own decoding region while being disjoint from those of all $\boldsymbol{s'} \neq \boldsymbol{s}$. This structural property enhanced key recovery robustness in Steps 5 and 5*.

VII. EXPERIMENTAL RESULTS

A. Hamming weight Classifier Evaluation

We employ the MLP model as the Hamming weight classifier in Steps 2 and 4 of the Secret Key Recovery Algorithm. As mentioned in Section V-C, the MLP model is trained on $\mathcal{T}_{\text{train}}$ and fine-tuned on $\mathcal{T}_{\text{fine-tune}}$, it takes subtrace $T[i]$ as input and outputs the corresponding $\text{HW}(r[i])$. This section evaluates the classifier's performance.

Table IV presents the performance of the model on the testset $\mathcal{T}_{\text{test}}^{\mathcal{P}_1}$, while Figure 4 displays the confusion matrix.

Our model achieved 97.11% classification accuracy, comparable to the model of identical architecture trained on $\mathcal{T}_{\text{train}}$ and tested on $\mathcal{T}_{\text{test}}^{\mathcal{P}_0}$. This result demonstrates that fine-tuning with limited data achieves equivalent performance to training with four times more data, significantly reducing training data requirements. Lower precision and recall rates for labels 0 and 11 stem from data skew (class imbalance), where these categories exhibit minimal sample counts. We attempted to mitigate this imbalance via resampling techniques, but observed no improvement.

TABLE IV
CLASSIFICATION REPORT

Class	Precision	Recall	F1-score	Support
0	0.8960	0.9234	0.9095	653
1	0.9654	0.9397	0.9524	7,334
2	0.9737	0.9591	0.9664	40,863
3	0.9746	0.9664	0.9705	135,553
4	0.9744	0.9703	0.9724	293,874
5	0.9715	0.9753	0.9734	448,640
6	0.9696	0.9769	0.9732	481,333
7	0.9705	0.9724	0.9714	366,013
8	0.9686	0.9651	0.9669	192,163
9	0.9690	0.9491	0.9589	66,817
10	0.9722	0.9189	0.9448	13,539
11	0.9533	0.8892	0.9201	1,218
Accuracy			0.9711	2,048,000
Macro Avg.	0.9632	0.9505	0.9567	2,048,000
Weighted Avg.	0.9711	0.9711	0.9711	2,048,000

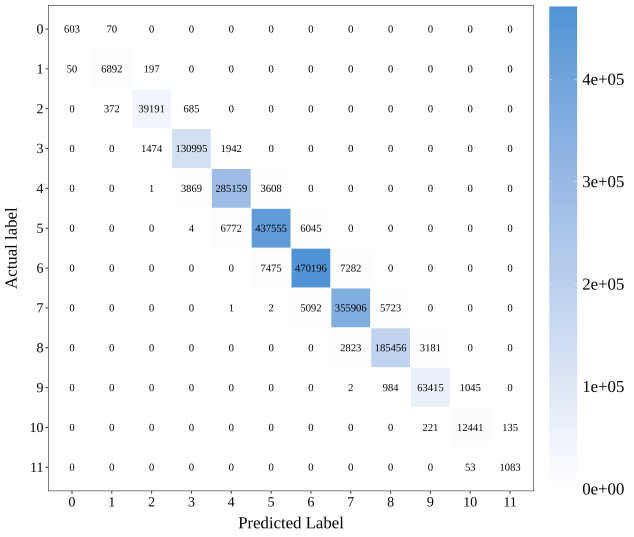


Fig. 4. Confusion Matrix

B. Secret Key Recovery Evaluation

The current classifier accuracy remains insufficient for complete Hamming weight recovery of a full-trace $\boldsymbol{T} = \{T[i] \mid i \in \{0, 1, \cdots, 255\}\}$, where $T[i]$ denotes the i-th subtrace. Given per-subtrace recovery accuracy p_{acc}, the probability of correctly recovering all 256 Hamming weights $\mathrm{HW}(r[i])$ is p_{acc}^{256}. In our experiment, the accuracy is 97.11%, yielding a 0.05% full-trace recovery probability, which is statistically negligible. Empirically, the probability of full-trace Hamming weight recovery on $\mathcal{T}_{\text{test}}^{\mathcal{P}_1}$ is only 0.162%, with an average of 7.39 erroneous coefficients per trace. Achieving a 97.47% full-trace recovery probability requires $p_{acc} \geq 99.99\%$. As a result, the $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ obtained in Step 5 of the Secret Key Recovery Algorithm contains errors at unknown positions and quantities, undermining recovery of secret coefficients, motivating the development of specialized error correction in Steps 2* and 5* of the Secret Key Recovery Algorithm.

Steps 2* and 5* are both based on the following observations:

1) The confusion matrix (Figure 4) reveals that misclassifications occur exclusively between adjacent classes; that is, subtrace $T[i]$ with true label $\mathrm{HW}(r[i])$ is only misclassified as either $\mathrm{HW}(r[i]) + 1$ or $\mathrm{HW}(r[i]) - 1$.
2) Analysis of the ranked Softmax output probabilities demonstrates that while the top-1 prediction accuracy is 97.11%, the top-2 cumulative accuracy approaches 100%.

These observations can be well explained by the fact that $r[i]$ with similar Hamming weights produce highly similar power consumption subtraces during processor operations, leading to their frequent confusion. This information significantly narrows the range of errors, allowing us to correct mistakes with minimal cost.

Before secret key recovery, we first evaluate the recovery of the Hamming weight distribution of $\boldsymbol{r}$ under the basic malicious ciphertext, which is executed in Step 2 and corrected in Step 2* of the Secret Key Recovery Algorithm. This is fundamental for accurately recovering the Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ and subsequently recovering the secret key. As mentioned in Step 2*, we first use the basic malicious ciphertext constructed from Row 1 parameters of Table III ($ku = 1586$, $kv[0] = 1040$) and repeat power consumption acquisition five times. Subsequently, we use the auxiliary basic malicious ciphertext constructed from Row 3 parameters ($ku = 1297$, $kv[0] = 1248$) and repeat power consumption acquisition another five times. By aggregating these **10** experimental results, we successfully recover the correct Hamming weight frequency distribution of $\boldsymbol{r}$ under the basic malicious ciphertext, validated over 1000 randomized trials.

After that, we evaluate the recovery of the Hamming weight distribution of $\boldsymbol{r}'$ under the derived malicious ciphertext and the Hamming weight distribution differential $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ which is executed in Steps 4 and 5 and corrected in Step 5*. Table V reports the success rates of $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ recovery using the malicious ciphertext constructed from Row 1 parameters of Table III ($ku = 1586$, $kv = [1040, 1873, 2497]$). Similarly, Table VI reports the success rates using the malicious ciphertext constructed from Row 2 parameters. The first column lists the secret key coefficient tuple $(s[i_1], s[i_2])$, the second column shows the corresponding $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ and the subsequent six columns report the recovery success rates across multiple repeated power consumption acquisitions. For example, Row 1 in Table V indicates that the recovery target $(s[i_1], s[i_2]) = (-2, -2)$ corresponds to $\Delta\boldsymbol{Dist}(\boldsymbol{s}) = (0, 0, 0, 0, 1, 0, 1, -2, 0, 0, 0, 0)$, the probability of directly recovering the correct $\Delta\boldsymbol{Dist}(\boldsymbol{s})$ without repetition is 81.33%, rising to 91.61% with two repetitions, and so on.

However, we observe exceptionally low recovery success rates for specific secret key coefficient tuples (marked in red in Tables V and VI), where success probabilities show no significant improvement even with repeated trace acquisitions. This phenomenon is attributed to the highly entangled error spaces of these tuples. We then develop a cascaded correction strategy, Step 5* of the Secret Key Recovery Algorithm, to address this issue. Briefly, we first repeat power consumption acquisitions and analysis until identifying the unique candidate in the united candidate set or reaching the M-acquisition limit

(we set M as 16), using malicious ciphertexts constructed from Row 1 parameters in Table III. If unsolved, we switch to Row 2 ciphertexts optimized for targets unsolved in Row 1 attacks, repeating acquisitions and analysis under identical termination conditions.

The above results demonstrate the performance of recovering one secret key coefficient tuple (containing two secret key coefficients). Based on these results, recovering a full secret key of Kyber-768 requires an average of $10+354\times3$ malicious ciphertexts with a success rate of 97.98% in our simulation. 10 is needed for the recovery of $\boldsymbol{Dist}(\mathrm{HW}(\boldsymbol{r}))$ under the basic malicious ciphertext, 354 is needed for the recovery of 256 secret key coefficients, and $\times3$ because the secret key in Kyber-768 contains $\texttt{KYBER\_K} = 3$ polynomials, each with 256 coefficients.

Compared to other attack schemes targeting Kyber/Saber with shuffling countermeasures: if we apply the bit-flip technique [18], [23] to Kyber-768, recovering the full secret key requires $3\times3\times257\times N$ traces (with $N = 10$ repetitions). Here, the first factor 3 corresponds to $\texttt{KYBER\_K} = 3$ polynomials per secret key; the second factor 3 arises because three message recoveries are needed to derive one secret key coefficient [29]; and 257 corresponds to 257 malicious ciphertexts. The method from [19], which first recovers the start/end indices of shuffling and then the corresponding secret key coefficients, requires 38,016 traces to recover the full secret key.

Our secret key recovery algorithm achieves 95.36% fewer malicious ciphertexts than the bit-flip technique while improving operational flexibility. These improvements originate from:

- **Parallel Recovery**: Our algorithm simultaneously recovers 2-3 secret key coefficients per operation, whereas bit-flip techniques recover only one message bit per operation.
- **Error Correction**: Our correction strategy effectively addresses the error in Hamming weight classifier, enabling reliable secret key recovery with minimal required trace capture repetitions.

VIII. Conclusion

The proposed attack efficiently extracts secret keys from Kyber-768 implementations protected with Fisher-Yates shuffled polynomial reduction, circumventing execution-order randomization defenses. It demonstrates that shuffling cannot eliminate higher-order statistical characteristics. Attackers can thus extract secrets by analyzing inter-trace statistical differentials. Crucially, our error correction method leverages the MLP classifier's systematic prediction features, enabling secret key recovery with minimal required trace capture repetitions under low-precision classifiers or low-SNR environments.

However, the attack's efficacy diminishes on hardware with parallelized arithmetic units, as concurrent operations obscure single-coefficient leakage. Future research should extend this framework to other lattice-based primitives and implementations.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[2] G. Alagic, G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller *et al.*, "Status report on the third round of the nist post-quantum cryptography standardization process," 2022.

[3] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A highly-efficient lattice-based post-quantum cryptography processor for IoT applications," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, pp. 130–153, 2024.

[4] X. Ji, J. Dong, J. Huang, Z. Yuan, W. Dai, F. Xiao, and J. Lin, "Eco-crystals: Efficient cryptography crystals on standard risc-v isa," *IEEE Transactions on Computers*, 2024.

[5] Z. Ye, J. Huang, T. Huang, Y. Bai, J. Li, H. Zhang, G. Li, D. Chen, R. C. Cheung, and K. Huang, "PQNTRU: Acceleration of NTRU-based schemes via customized post-quantum processor," *IEEE Transactions on Computers*, 2025.

[6] F. Antognazza, A. Barenghi, and G. Pelosi, "An efficient and unified rtl accelerator design for hqc-128, hqc-192, and hqc-256," *IEEE Transactions on Computers*, 2025.

[7] C. Wang, Z. Ye, H. Shen, and K. Huang, "A folded computation-in-memory accelerator for fast polynomial multiplication in BIKE," in *European Conference on Parallel Processing*. Springer, 2024, pp. 137–151.

[8] J. Lopez-Valdivieso and R. Cumplido, "Design and implementation of hardware-software architecture based on hashes for sphincs+," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 4, pp. 1–22, 2024.

[9] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1996, pp. 104–113.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer, 1999, pp. 388–397.

[11] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*. Springer, 2001, pp. 200–210.

[12] C. Percival, "Cache missing for fun and profit," 2005.

[13] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17*. Springer, 1997, pp. 513–525.

[14] B. Hettwer, S. Gehrer, and T. Güneysu, "Applications of machine learning techniques in side-channel attacks: a survey," *Journal of Cryptographic Engineering*, vol. 10, no. 2, pp. 135–162, 2020.

[15] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable sca countermeasures against single trace attacks for the ntt: A performance evaluation study over kyber and dilithium on the arm cortex-m4," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2020, pp. 123–146.

[16] D. Xu, K. Wang, and J. Tian, "A hardware-friendly shuffling countermeasure against side-channel attacks for kyber," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2025.

[17] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked ind-cca secure saber kem implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 676–707, 2021.

[18] P. Ravi, S. Bhasin, S. S. Roy, and A. Chattopadhyay, "On exploiting message leakage in (few) nist pqc candidates for practical message recovery attacks," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 684–699, 2021.

[19] L. Backlund, K. Ngo, J. Gärtner, and E. Dubrova, "Secret key recovery attack on masked and shuffled implementations of crystals-kyber and saber," in *International Conference on Applied Cryptography and Network Security*. Springer, 2023, pp. 159–177.

[20] B. Udvarhelyi, O. Bronchain, and F.-X. Standaert, "Security analysis of deterministic re-keying with masking and shuffling: Application to isap," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2021, pp. 168–183.

[21] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, "Adapting belief propagation to counter shuffling of ntts," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 60–88, 2023.

[22] S. Jendral, K. Ngo, R. Wang, and E. Dubrova, "A single-trace message recovery attack on a masked and shuffled implementation of crystals-kyber," *Cryptology ePrint Archive*, 2023.

TABLE V
MAPPING TABLE AND REPEAT TIMES/SUCCESS RATE WITH $ku = 1586, kv = [1040, 1873, 2497]$

$\boldsymbol{s} = (s[i_1], s[i_2])$	$\Delta \boldsymbol{Dist}(\boldsymbol{s})$	Repeat times/Success Rate					
		1	2	3	4	5	6
$(-2,-2)$	$(0,0,0,0,0,0,-2,1,0,1,0,0)$	81.33%	91.61%	98.87%	99.39%	99.90%	99.97%
$(-2,-1)$	$(0,0,0,0,0,0,-1,1,0,0,0,0)$	0.00%	0.38%	0.80%	0.66%	1.07%	1.32%
$(-2,0)$	$(0,0,-1,0,0,1,-1,0,0,1,0,0)$	92.34%	98.09%	99.78%	99.91%	99.97%	100.00%
$(-2,1)$	$(0,0,0,0,-1,0,0,0,0,1,0,0)$	95.00%	94.88%	99.45%	99.75%	99.98%	99.97%
$(-2,2)$	$(0,0,0,0,1,0,-1,-1,0,1,0,0)$	96.09%	98.01%	99.88%	99.96%	100.00%	100.00%
$(-1,-2)$	$(0,0,0,0,0,0,0,1,0,-1,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(-1,-1)$	$(0,0,0,0,0,0,1,1,0,-2,0,0)$	95.55%	99.79%	99.98%	100.00%	100.00%	100.00%
$(-1,0)$	$(0,0,-1,0,0,1,1,0,0,-1,0,0)$	42.89%	67.20%	80.16%	88.96%	92.82%	95.70%
$(-1,1)$	$(0,0,0,0,-1,0,2,0,0,-1,0,0)$	94.53%	94.08%	99.08%	99.40%	99.92%	99.96%
$(-1,2)$	$(0,0,0,0,1,0,1,-1,0,-1,0,0)$	88.20%	97.92%	99.72%	99.97%	99.98%	100.00%
$(0,-2)$	$(0,0,-1,0,0,0,0,1,0,0,0,0)$	34.61%	54.49%	70.72%	81.31%	87.93%	92.09%
$(0,-1)$	$(0,0,-1,0,0,0,1,1,0,-1,0,0)$	38.20%	61.30%	76.25%	85.81%	91.63%	94.49%
$(0,0)$	$(0,0,-2,0,0,1,1,0,0,0,0,0)$	94.38%	99.50%	99.98%	99.99%	100.00%	100.00%
$(0,1)$	$(0,0,-1,0,-1,0,2,0,0,0,0,0)$	95.47%	95.89%	99.67%	99.81%	99.98%	100.00%
$(0,2)$	$(0,0,-1,0,1,0,1,-1,0,0,0,0)$	10.94%	20.75%	29.07%	36.66%	43.62%	49.58%
$(1,-2)$	$(0,0,1,0,-1,0,-1,1,0,0,0,0)$	84.53%	94.54%	99.38%	99.78%	99.98%	100.00%
$(1,-1)$	$(0,0,1,0,-1,0,0,1,0,-1,0,0)$	94.61%	96.50%	99.59%	99.91%	100.00%	99.99%
$(1,0)$	$(0,0,0,0,-1,1,0,0,0,0,0,0)$	0.00%	42.04%	68.39%	82.05%	90.17%	94.63%
$(1,1)$	$(0,0,1,0,-2,0,1,0,0,0,0,0)$	93.98%	96.49%	99.53%	99.86%	99.96%	100.00%
$(1,2)$	$(0,0,1,0,0,0,0,-1,0,0,0,0)$	79.22%	92.37%	97.90%	98.98%	99.70%	99.84%
$(2,-2)$	$(0,0,0,0,0,0,0,0,0,0,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(2,-1)$	$(0,0,0,0,0,0,1,0,0,-1,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(2,0)$	$(0,0,-1,0,0,1,1,-1,0,0,0,0)$	0.08%	23.34%	46.16%	62.43%	74.37%	82.85%
$(2,1)$	$(0,0,0,0,-1,0,2,-1,0,0,0,0)$	29.38%	53.87%	71.18%	80.98%	87.18%	91.20%
$(2,2)$	$(0,0,0,0,1,0,1,-2,0,0,0,0)$	75.00%	93.60%	98.48%	99.67%	99.94%	99.98%

TABLE VI
MAPPING TABLE AND REPEAT TIMES/SUCCESS RATE WITH $ku = 1743, kv = [1040, 1873, 2497]$

$\boldsymbol{s} = (s[i_1], s[i_2])$	$\Delta \boldsymbol{Dist}(\boldsymbol{s})$	Repeat times/Success Rate					
		1	2	3	4	5	6
$(-2,-2)$	$(0,0,0,0,1,0,1,-2,0,0,0,0)$	75.39%	92.86%	98.61%	99.69%	99.98%	100.00%
$(-2,-1)$	$(0,0,0,0,-1,0,2,-1,0,0,0,0)$	29.06%	53.50%	70.86%	79.48%	85.91%	89.39%
$(-2,0)$	$(0,0,-1,0,0,1,1,-1,0,0,0,0)$	0.00%	25.23%	49.04%	67.10%	77.46%	84.94%
$(-2,1)$	$(0,0,0,0,0,0,1,0,0,-1,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(-2,2)$	$(0,0,0,0,0,0,0,0,0,0,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(-1,-2)$	$(0,0,1,0,0,0,0,-1,0,0,0,0)$	76.64%	90.09%	96.75%	98.81%	99.54%	99.76%
$(-1,-1)$	$(0,0,1,0,-2,0,1,0,0,0,0,0)$	95.62%	97.74%	99.78%	99.92%	100.00%	100.00%
$(-1,0)$	$(0,0,0,0,-1,1,0,0,0,0,0,0)$	0.00%	39.80%	65.11%	79.59%	87.38%	92.78%
$(-1,1)$	$(0,0,1,0,-1,0,0,1,0,-1,0,0)$	94.69%	96.14%	99.44%	99.72%	99.95%	99.99%
$(-1,2)$	$(0,0,1,0,-1,0,-1,1,0,0,0,0)$	82.81%	92.00%	98.80%	99.62%	99.88%	99.95%
$(0,-2)$	$(0,0,-1,0,1,0,1,-1,0,0,0,0)$	8.98%	16.58%	24.47%	30.06%	36.78%	42.81%
$(0,-1)$	$(0,0,-1,0,-1,0,2,0,0,0,0,0)$	95.31%	96.29%	99.67%	99.80%	99.99%	99.98%
$(0,0)$	$(0,0,-2,0,0,1,1,0,0,0,0,0)$	94.30%	99.72%	99.99%	100.00%	100.00%	100.00%
$(0,1)$	$(0,0,-1,0,0,0,1,1,0,-1,0,0)$	38.91%	62.64%	76.67%	86.17%	91.61%	94.90%
$(0,2)$	$(0,0,-1,0,0,0,0,1,0,0,0,0)$	37.50%	56.71%	75.09%	84.19%	90.07%	94.41%
$(1,-2)$	$(0,0,0,0,1,0,1,-1,0,-1,0,0)$	89.77%	98.67%	99.88%	99.98%	99.99%	100.00%
$(1,-1)$	$(0,0,0,0,-1,0,2,0,0,-1,0,0)$	94.84%	94.34%	99.48%	99.47%	99.93%	99.96%
$(1,0)$	$(0,0,-1,0,0,1,1,0,0,-1,0,0)$	44.06%	68.60%	83.23%	90.13%	94.29%	96.87%
$(1,1)$	$(0,0,0,0,0,0,1,1,0,-2,0,0)$	95.16%	99.79%	99.99%	100.00%	100.00%	100.00%
$(1,2)$	$(0,0,0,0,0,0,0,1,0,-1,0,0)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(2,-2)$	$(0,0,0,0,1,0,-1,-1,0,1,0,0)$	94.45%	97.80%	99.70%	99.96%	99.99%	100.00%
$(2,-1)$	$(0,0,0,0,-1,0,0,0,0,1,0,0)$	93.83%	93.58%	99.28%	99.63%	99.94%	99.97%
$(2,0)$	$(0,0,-1,0,0,1,-1,0,0,1,0,0)$	92.03%	98.16%	99.78%	99.98%	100.00%	100.00%
$(2,1)$	$(0,0,0,0,0,0,-1,1,0,0,0,0)$	0.00%	0.24%	0.37%	0.71%	0.52%	0.96%
$(2,2)$	$(0,0,0,0,0,0,-2,1,0,1,0,0)$	83.12%	90.87%	98.74%	99.36%	99.91%	99.95%

[23] K. Ngo, E. Dubrova, and T. Johansson, "A side-channel attack on a masked and shuffled software implementation of saber," *Journal of Cryptographic Engineering*, vol. 13, no. 4, pp. 443–460, 2023.

[24] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber (version 3.02) – submission to round 3 of the nist post-quantum project," aug 2021. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

[25] R. A. Fisher and F. Yates, "Statistical tables for biological, agricultural and medical research." 1963.

[26] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer, 2004, pp. 16–29.

[27] B. L. Welch, "The generalization of 'student's'problem when several different population varlances are involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[28] Y. Ji and E. Dubrova, "A side-channel attack on a masked hardware implementation of crystals-kyber," in *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*, 2023, pp. 27–37.

[29] J. Wang, W. Cao, H. Chen, and H. Li, "Practical side-channel attack on message encoding in masked kyber," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2022, pp. 882–889.