# Profiling-Device-Free SASCA Framework for ML-KEM

Yuxuan Wang
Shanghai Jiao Tong University
Shanghai, China
18588297218@sjtu.edu.cn

## Abstract

In side-channel analysis of ML-KEM (a NIST-standard PQC algorithm), SASCA is a powerful profiling attack. However, obtaining a profiling device strictly matching the target is challenging in practice. To address this, we propose the first profiling-device-free SASCA framework for ML-KEM. The framework first controls the NTT input by choosing ciphertexts and trains a leakage model. Subsequently, leveraging the similarity between NTT and INTT, it uses adversarial unsupervised domain adaptation to fine-tune the model for INTT and recover its secret input. Validated on real embedded devices, the framework achieves effective key recovery using a comparable number of traces to profiling SASCA.

## Keywords

ML-KEM, Side-Channel Attacks, Transfer Learning, SASCA, Adversarial Networks

## 1 Introduction

In response to the growing threat posed by increasingly powerful quantum computers to existing public-key cryptosystems, the National Institute of Standards and Technology (NIST) has standardized several Post-Quantum Cryptography (PQC) algorithms that are resistant to both classical and quantum attacks. Among them, ML-KEM [22] (the standardized version of CRYSTALS-Kyber) is a lattice-based Key Encapsulation Mechanism (KEM) with relatively high computational efficiency, making it well-suited for deployment in resource-constrained embedded devices. Embedded devices are frequently deployed in scenarios where attackers have physical access, underscoring the need for ML-KEM implementations to be resilient against physical attacks like Side-Channel Analysis (SCA). Therefore, SCA on ML-KEM implementations has attracted considerable attention from the research community [16, 21, 28, 31].

In SCA targeting ML-KEM, Soft Analytical Side-Channel Attack (SASCA) [30] represents a powerful class of profiling attacks. SASCA exploits the leakage of multiple related sensitive intermediates to recover a subset of them (the target intermediates). In short, SASCA first uses Template Attack (TA) [5] to extract the probability distributions of each intermediate, then constructs a factor graph based on the relationships among the intermediates, and finally performs probabilistic inference on the graph using the Belief Propagation (BP) [33] algorithm to compute the marginal distributions of the targets. By leveraging more leakage information, SASCA can recover the key with only a small number of traces—or even a single trace.

Most studies on SASCA for ML-KEM typically target the Number Theoretic Transform (NTT). The NTT is an FFT-like algorithm over $\mathbb{Z}_q$, widely used to accelerate multiplication in the polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$ found in lattice-based cryptography like ML-KEM. SASCA exploits the leakage of the inputs and outputs of all basic blocks (butterfly units) of the NTT to recover its input.

Primas et al. [25] proposed the first SASCA targeting the NTT component of ML-KEM. This method achieved single-trace attacks on low-noise datasets and was capable of compromising masked implementations, but required constructing 983,168 templates for modular multiplication operations. Pessl et al. [23] shifted to building templates for load and store operations, thereby significantly reducing the number of templates required. They further refined the factor graph structure by merging nodes and introducing a damping mechanism to optimize BP convergence, thereby allowing their method to tolerate Gaussian noise with a standard deviation of 1.3. Assael et al. [1] designed an optimized factor graph for an NTT implementation tailored to the ARM Cortex-M4, utilizing leakage from additional intermediates. Hamburg et al. [15] proposed an approach that uses chosen ciphertexts to construct sparse NTT inputs, thus increasing the noise tolerance to 2.7. For shuffled NTT implementations, Hermelink et al. [17] incorporated "shuffle nodes" into the factor graph to defeat fine shuffling and developed a matching algorithm to counter coarse shuffling.

Although previous research has established SASCA as one of the most powerful SCA methods against ML-KEM, it still relies on the assumption of a profiling device—one fully controlled by the attacker and nearly identical to the target device. Many existing studies employ a single device for both profiling and attacking in controlled laboratory settings. However, this approach faces practical limitations in real-world scenarios. First, attackers often need to perform a teardown and PCB analysis to identify the chip model used in the target device [26, 27]. Second, even with knowledge of the chip model, obtaining identical PCBs is challenging. Furthermore, even among devices of the same model, minor hardware variations, environmental noise, or differences in acquisition settings can introduce deviations that lead to analysis failures—a challenge commonly referred to as the portability issue [4, 10].

Therefore, this paper aims to design a profiling-device-free SASCA framework for ML-KEM, eliminating the need for a profiling device while retaining a strong capability to exploit leakage from multiple intermediates to achieve few-trace attacks.

To accomplish this objective, we consider constructing leakage models on the target device instead of a profiling device. We observe that during the decapsulation phase of ML-KEM, the target device

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*Conference'17, Washington, DC, USA*
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

sequentially executes NTT($u$) and INTT($\hat{s}^T \circ \hat{u}$), where $u$ is a polynomial vector derived from the ciphertext, $s$ denotes the private key, and $\hat{u}$ and $\hat{s}$ represent their respective NTT-transformed forms. By choosing ciphertexts, we can construct leakage models for NTT($u$) with non-sensitive input. Given the similarities between NTT and INTT in their iterative algorithmic structure and specific operation types, we find that NTT leakage models can be fine-tuned via Unsupervised Domain Adaptation (UDA) to analyze INTT leakage, thereby recovering the sensitive input $\hat{s}^T \circ \hat{u}$ and ultimately the private key. Note that although ML-KEM employs the Fujisaki-Okamoto (FO) transform [7] and is considered CCA2-secure, this does not prevent chosen-ciphertext SCAs. This is because our target NTT/INTT instances are executed before FO's re-encryption and comparison (which rejects invalid ciphertexts).

In summary, the main contributions of this work are:

- We propose the first profiling-device-free SASCA framework for ML-KEM. Compared to existing profiling SASCAs, it requires no profiling device, enabling attacks under relaxed assumptions. Compared to non-profiling attacks like CPA [2], it leverages leakage of more intermediates. Furthermore, unlike other non-profiling methods requiring fixed keys for all traces [29, 34], our framework only needs a few attack traces to share a key (not required in profiling), making it robust against key updates.
- We introduce Transfer Learning (TL) into SASCA. First, we use neural networks instead of TA templates for probability estimation. Second, we attack sensitive operations by fine-tuning the model trained for non-sensitive operations using adversarial UDA techniques. Unlike prior works that focus on learning device-invariant features[3, 4], we demonstrate the novel capability of TL to extract operation-invariant features.
- Our framework was validated on a real Atmel SAM4S embedded device, recovering the full key with only 100 profiling and 6.6 attack traces (average). This is slightly more than SASCA (100 profiling + 2.6 attack traces) but far fewer than the non-profiling method CPA, which requires over 400 traces.

In Section 2, we provide preliminaries. Section 3 describes the proposed profiling-device-free SASCA framework. Section 4 presents the experimental setup and results. Section 5 discusses parameter selection and countermeasures. Section 6 concludes the paper.

# 2 Preliminaries

## 2.1 ML-KEM

ML-KEM is a NIST-standardized key encapsulation mechanism based on the Module Learning With Errors (M-LWE) problem [18], comprising three phases: key generation, encapsulation, and decapsulation. Its core is a public-key encryption algorithm called K-PKE. The computations in ML-KEM are performed over the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$, where $\mathbb{Z}_q$ is the ring of integers modulo $q = 3329$. This work focuses on ML-KEM decapsulation, whose core is K-PKE decryption (Algorithm 1). The private key $\hat{s}$ is the NTT form of a polynomial vector $\mathbf{s}$ of length $k$, whose coefficients belong to a small interval $[-\eta_1, \eta_1]$ (e.g., $\eta_1 = 2$ for ML-KEM 512).

**Algorithm 1** K-PKE Decryption (Simplified)

**Input:** Private key $\hat{s} \in R_q^k$; Ciphertext $(\mathbf{u}, v) \in R_q^k \times R_q$
**Output:** Message $m \in \{0,1\}^{256}$
1: $w \leftarrow v - \text{INTT}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u}))$
2: **return** $m = \lceil (2/q)w \rfloor$

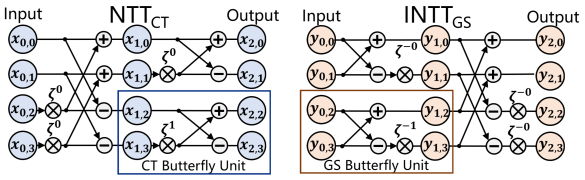



**Figure 1: 4-Point NTT and INTT with CT/GS Butterflies.**

## 2.2 NTT and INTT

NTT, whose inverse is INTT, is an analogue of FFT over $\mathbb{Z}_q$. NTT is used to reduce the computational complexity of polynomial multiplication in $R_q$ from $O(n^2)$ to $O(n \log n)$. The accelerated multiplication is computed as $f \cdot g = \text{INTT}(\text{NTT}(f) \circ \text{NTT}(g))$, where "$\circ$" denotes component-wise multiplication. As shown in Figure 1, both NTT and INTT follow a divide-and-conquer iterative structure similar to FFT, with the butterfly unit as the core component. Typically, NTT uses the Cooley-Tukey (CT) butterfly [6], while INTT employs the Gentleman-Sande (GS) butterfly [8]:

$$\text{Butterfly}_{CT}(x_1, x_2) = ((x_1 + \omega \cdot x_2) \bmod q, (x_1 - \omega \cdot x_2) \bmod q), \quad (1)$$

$$\text{Butterfly}_{GS}(y_1, y_2) = ((y_1 + y_2) \bmod q, ((y_1 - y_2) \cdot \omega^{-1}) \bmod q). \quad (2)$$

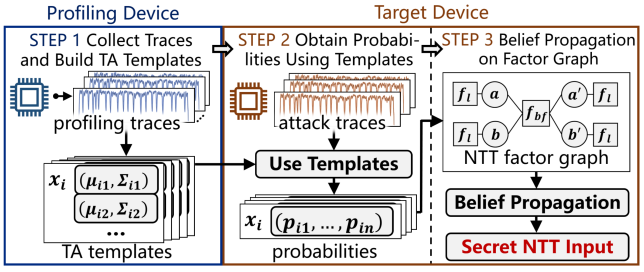



**Figure 2: SASCA on NTT.**

## 2.3 SASCA

Figure 2 shows the SASCA on NTT, following a TA+BP workflow. First, traces from the profiling device are used to build TA templates for intermediates $(x_1, x_2, \ldots, x_m)$, such as all circle-marked variables in Figure 1. Each template $(\mu_{ij}, \Sigma_{ij})$ models leakage with a multivariate Gaussian distribution per class (e.g., Hamming weight) of $x_i$. Second, a few attack traces are acquired from the target device, and the templates compute probability distributions $P = (p_{i1}, \ldots, p_{in})$ for each $x_i$. Third, a factor graph is built for NTT, and BP is executed to recover the secret input. For details of BP, see Appendix A of [15]. Step 3 in Figure 2 shows the factor graph of CT-Butterfly. The complete NTT graph is formed by connecting multiple butterfly graphs according to the NTT structure. Here, the circular variable

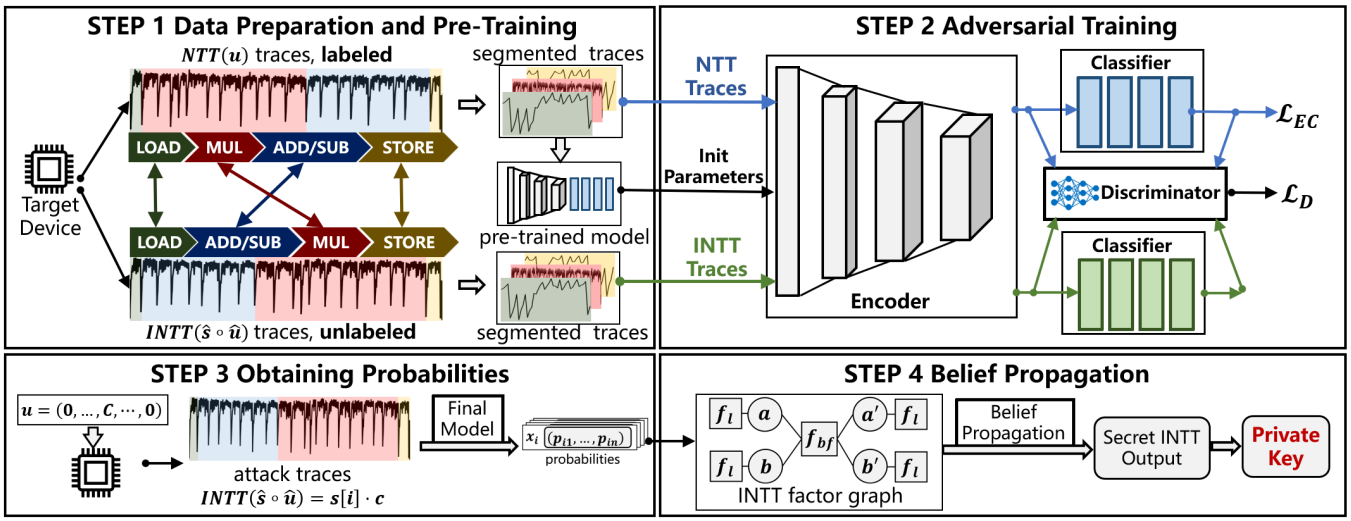



**Figure 3: The Profiling-Device-Free SASCA Framework.**

nodes represent marginal distributions, while the square factor nodes represent constraints. There are two types of factor nodes: $f_l = Pr(x = i|l)$ represents the probability distribution of intermediate $x$ given leakage $l$, which is obtained by TA; $f_{bf}$ represents the relationship between the inputs $(a, b)$ and outputs $(a', b')$:

$$f_{bf}(a, b, a', b') = \begin{cases} 1 & \text{if } (a', b') = \text{Butterfly}_{CT}(a, b) \\ 0 & \text{otherwise} \end{cases}. \quad (3)$$

## 3 Profiling-Device-Free SASCA Framework

The overall architecture of our proposed profiling-device-free SASCA framework is illustrated in Figure 3. The entire procedure comprises four steps. Briefly, Step 1 involves pre-training a neural network model on a labeled set of NTT traces. Step 2 introduces an unlabeled set of INTT traces for adversarial training to generate the final model. In Step 3, ciphertexts are selected to generate INTT instances with narrowly distributed secret outputs. We collect their traces and derive the probability distributions of intermediates. Step 4 performs the BP algorithm on a factor graph to recover the secret input of the INTT and ultimately restore the key. The technical details of each step will be elaborated in the following subsections.

### 3.1 Data Preparation and Pre-Training

We assume the attacker can invoke the ML-KEM decapsulation function deployed on the target device and collect side-channel traces. By inputting ciphertexts $u$, traces corresponding to the sequential execution of NTT$(u)$ and INTT$(\hat{s}^T \circ \hat{u})$ are captured. Given that both NTT and INTT are composed of repeated butterfly operations, the framework constructs datasets with butterfly operations as the fundamental trace blocks—generating a labeled NTT dataset $\mathcal{T}$ and an unlabeled INTT dataset $\mathcal{T}'$.

Here, we discuss the similarity between $\mathcal{T}$ and $\mathcal{T}'$. For an instance with input $(a, b)$ and output $(a', b')$, the CT-Butterfly sequentially computes in $\mathbb{Z}_q$:

$$t = b \cdot \omega,\ a' = a + t,\ b' = a - t, \quad (4)$$

while the GS-Butterfly computes:

$$a' = a + b,\ t = a - b,\ b' = t \cdot \omega^{-1}. \quad (5)$$

Both computation streams consist of modular multiplication, addition, and subtraction. Taking into account the load and store operations for inputs and outputs, the corresponding relationship in Step 1 of Figure 3 is established. By comparing the traces of the two butterflies, we can determine the operation corresponding to each segment of the trace and generate a segmented trace dataset.

For each leaking trace segment and its intermediate value labels, the framework establishes a model (the pre-trained model). Unlike existing SASCA works that employ TA templates, we use Convolutional Neural Networks (CNNs) to model the leakage. On one hand, this makes our model compatible with transfer learning techniques. On the other hand, neural networks eliminate the need for manual pre-processing and perform better on real-world traces with non-Gaussian noise [14, 19, 24].

### 3.2 Adversarial Training

While NTT and INTT share similarities, their leakage patterns remain distinct, thereby limiting the effectiveness of directly applying pre-trained models for attacks. For load/store operations, these differences arise from variations in source/destination addresses and registers. For arithmetic operations, the discrepancies arise from different operands (e.g., NTT uses multiplier $a$ vs. $a - b$ in INTT).

Therefore, it is necessary to fine-tune the pre-trained model on INTT traces. We adopt a design philosophy similar to that of Generative Adversarial Networks (GANs) [11]. GAN consists of a generator *Gen* and a discriminator *Dis*. It aims to make the samples generated by *Gen* indistinguishable from real data to *Dis*. In SCA, Cao et al. [4] used adversarial training to learn common features of the same operation across devices of the same model. This paper applies GAN-like networks to address another problem: learning common features across similar operations on the same device.

We aim to fine-tune a pre-trained model using unlabeled INTT traces to achieve satisfactory classification performance on both

NTT (source domain) and INTT (target domain) traces. This is a task known as unsupervised domain adaptation. As shown in Step 2 of Figure 3, our network consists of:

- Encoder (*Enc*) extracts a feature vector $\mathbf{f}$. It captures common features while aligning source and target domains. Here, we employ the convolutional layers of the pre-trained CNN as *Enc*.
- Classifier (*Cls*) corresponds to the fully connected layers of the CNN. It maps $\mathbf{f}$ to logits $\mathbf{y}$, aiming to match ground-truth labels.
- Discriminator (*Dis*) judges whether the input belongs to the source or target domain based on the internal states of the neural network. We employ a Multilayer Perceptron (MLP) network as Dis and use the outer product $\mathbf{f}\times\mathbf{y}$ as its input to comprehensively characterize the internal state.

The training objective of each component is achieved by designing an adversarial loss function. For the discriminator *Dis*, the loss function $\mathcal{L}_D$ is designed such that for data from the source domain $X_{NTT}$, the corresponding $\mathbf{f}_{NTT}\times\mathbf{y}_{NTT}$ is classified as 1, while $\mathbf{f}_{INTT}\times\mathbf{y}_{INTT}$ is classified as 0 for data from $X_{INTT}$:

$$\begin{aligned}\mathcal{L}_D(X_{NTT}, X_{INTT}) = &-\mathbb{E}_{x\sim X_{NTT}}\log\left(Dis(\mathbf{f}_{NTT}\times\mathbf{y}_{NTT})\right)\\ &-\mathbb{E}_{x\sim X_{INTT}}\log(1-Dis(\mathbf{f}_{INTT}\times\mathbf{y}_{INTT})).\end{aligned}\tag{6}$$

To ensure that the fine-tuned model retains both classification ability and operational invariance, the loss function $\mathcal{L}_{EC}$ for *Enc* and *Cls* consists of two parts. The first term uses Cross-Entropy (CE) loss to quantify the discrepancy between the prediction $\mathbf{y}$ and the label $\mathbf{l}$ (from the set $L_{NTT}$) in the source domain; the second term is an adversarial loss, designed to make $\mathbf{f}\times\mathbf{y}$ generated by *Enc* and *Cls* as indistinguishable as possible for *Dis*. A parameter $\lambda$ is used to balance the two terms. $\mathcal{L}_{EC}$ is defined as:

$$\begin{aligned}\mathcal{L}_{EC}(X_{NTT}, X_{INTT},\ L_{NTT}) = &\ CE(\mathbf{y}_{NTT}, \mathbf{l}_{INTT})\\ &-\lambda\cdot\mathbb{E}_{x\sim X_{INTT}}\log(Dis(\mathbf{f}_{INTT}\times\mathbf{y}_{INTT})).\end{aligned}\tag{7}$$

The network is trained as follows: The parameters of *Enc* and *Cls* are initialized using the pre-trained model. As shown in Figure 3, each iteration feeds *Enc* with one batch of labeled NTT traces and one batch of unlabeled INTT traces, producing classification outputs for NTT and feature vectors $\mathbf{f}_{NTT}\times\mathbf{y}_{NTT}$ and $\mathbf{f}_{INTT}\times\mathbf{y}_{INTT}$. *Dis* is trained first on the feature vectors by minimizing $\mathcal{L}_D$, followed by *Enc* and *Cls* by minimizing $\mathcal{L}_{EC}$. After the adversarial iterations, the resulting *Enc* and *Cls* form the final operation-invariant model.

## 3.3 Obtaining Probabilities

The third step acquires a small number of INTT traces (attack traces) that share the same secret key, using the final model's output vector as the probabilities of each intermediate.

Here, we focus on the strategy for choosing ciphertexts. Both ciphertext $u$ and the private key $s$ are polynomial vectors of length $k$ (a ML-KEM parameter, e.g. $k=2$ for ML-KEM 512). Therefore, the output of the target INTT operation can be expressed as:

$$\mathrm{INTT}(\hat{s}^T\circ\hat{u}) = s\cdot u = \sum_{i=0}^{k-1} s[i]\cdot u[i].\tag{8}$$

Our method recovers each polynomial of $s$ sequentially. By zeroing all but the $i$-th polynomial of $u$, the INTT output becomes $s[i]\cdot u[i]$, enabling the recovery of $s[i]$ using the known $u[i]$. Furthermore, literature indicates that SASCA can tolerate higher noise when the intermediates have a smaller range [23]. Therefore, we construct an INTT instance with a narrowly distributed output by choosing ciphertexts. Specifically, we take $u[i]=(0,0,...,c)$, meaning only the constant term is set to a constant $c$. In this case, the INTT output becomes $s[i]\cdot c$. Since the coefficients of $s$ lie in the small set $\{-\eta_1,\ -\eta_1+1,\ ...,\ \eta_1\}$, the INTT output coefficients fall within $\{-c\cdot\eta_1,\ c\cdot(-\eta_1+1),\ ...,\ c\cdot\eta_1\}$. For INTT, the output of the final layer GS-Butterflies are multiplied by 3303 ($128^{-1}$ *mod* 3329) to produce the final output, thus the butterfly outputs fall within $\{-128\cdot c\cdot\eta_1,\ 128\cdot c\cdot(-\eta_1+1),\ ...,\ 128\cdot c\cdot\eta_1\}$. For ML-KEM 512 ($\eta_1$=2), this yields only 5 possible values. Note that $c$ cannot be an arbitrary element of $\mathbb{Z}_q$ because ML-KEM compresses and decompresses the coefficients of $u$ to save space; according to the decompression algorithm, $c$ is subject to the following constraints:

$$c\in\left\{x\in\mathbb{Z}_q\mid\exists y\in\mathbb{Z}_{2^d},\ x=\left\lceil\frac{q}{2^d}\cdot y\right\rfloor\right\}.\tag{9}$$

## 3.4 Key Recovery

The final step of the framework leverages the leakage from all intermediates to recover the secret INTT output $s[i]\cdot c$, thereby enabling the recovery of each element $s[i]$ and ultimately the recovery of the private key. Specifically, by applying the corresponding final models to the trace segments of different operations in each GS-Butterfly, we obtain probability distributions for the butterfly inputs and outputs ($a$, $b$, $a'$, and $b'$). As illustrated in Figure 3, similar to existing SASCA methods, the probability distributions of intermediates are embedded into the butterfly factor graph as factor nodes $f_l$. These butterfly graphs are then linked according to the INTT structure to form the complete INTT factor graph. Finally, performing BP on this factor graph yields the marginal distributions of the 256 INTT output intermediates.

# 4 Experiments on a Real Device

## 4.1 Setup and Dataset

A typical application of post-quantum cryptographic algorithms is their deployment on embedded devices. Therefore, we selected an Atmel SAM4S microcontroller based on the ARM Cortex-M4 architecture as the target device. A ChipWhisperer Husky was used for power trace acquisition, operating at a sampling frequency of 29.45 MHz, while the target device ran at 7.37 MHz. Following NIST's official pseudocode, we implemented a standard CT-Butterfly-based NTT and a GS-Butterfly-based INTT on the target device. Although different implementations of NTT/INTT may exhibit slight variations, they generally share a similar underlying structure, to which our framework remains readily applicable.

The proposed framework is validated on the ML-KEM variant ML-KEM 512. The dataset comprises profiling, transfer, and attack traces. First, we run the ML-KEM decapsulation algorithm with a random ciphertext $u$ collecting 100 side-channel traces each for the $\mathrm{NTT}(u)$ and $\mathrm{INTT}(\hat{s}^T\circ\hat{u})$ operations in each execution—serving as labeled profiling and unlabeled transfer traces, respectively. Each NTT/INTT trace consists of 896 butterfly segments, each comprising 800 points. This phase does not require fixed ciphertexts and remains unaffected by key updates. Next, a small set of INTT traces with fixed keys is collected following the ciphertext choosing strategy in Section 3.3, to generate attack traces. For each key, 10 attack

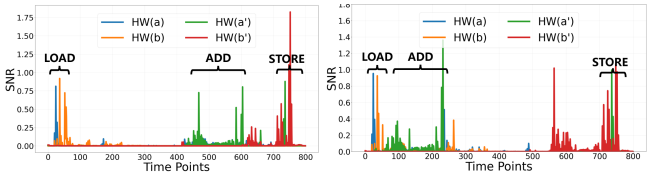



**Figure 4: SNR of NTT Traces (Left) and INTT Traces (Right).**

| Operation | NTT Intermediate | INTT Intermediate |
|---|---|---|
| LOAD | NTT input $a$ | INTT input $a$ |
| LOAD | NTT input $b$ | INTT input $b$ |
| ADD | NTT output $a' = a + \omega b$ | INTT output $a' = a + b$ |
| STORE | NTT output $b'$ | INTT output $b'$ |

**Table 1: Target Operations.**

traces are obtained: 5 for recovering each of the two polynomials of $s$. The attack dataset includes data of 10 distinct keys to evaluate the performance.

Figure 4 shows the Signal-to-Noise Ratio (SNR) of the traces of the butterflies in NTT and INTT for inputs $(a, b)$ and outputs $(a', b')$. The SNR curves indicate that NTT and INTT exhibit similar leakage in the three operations of LOAD, ADD, and STORE. Thus, we select these operations as the target segments and train four networks. Table 1 lists the variables corresponding to the labels (NTT intermediates) and recovery targets (INTT intermediates). This paper uses the Hamming weight of the variables as the labels.

First, the experiment validates that CNN model can effectively learn NTT leakage features (Subsection 4.2). Subsequently, it shows that the model trained on NTT cannot directly attack INTT, whereas the adversarially trained model can effectively learn operation-invariant features, thus enabling the analysis of INTT leakage (Subsection 4.3). Finally, the evaluation of the key recovery capability of the proposed framework is presented (Subsection 4.4). The experimental code is available in the following repository: https://anonymous.4open.science/r/pdfSASCA-1342.

## 4.2 Effectiveness of the Pre-Training Model

Existing SASCA research on ML-KEM uses TA to analyze intermediate leakage. To extract operation-invariant features via transfer learning, this work replaces TA with neural networks. Specifically, the pre-trained CNN comprises four convolutional layers, three fully connected layers, and dropout to prevent overfitting. Figure 5 compares CNN and TA performance in analyzing the LOAD($a$) operation in NTT: the x-axis denotes the number of Butterfly trace segments for profiling, and the y-axis represents validation accuracy. Since TA is sensitive to Point of Interest (PoI) selection, three common methods—DOM [5], SNR [20], and SOSD [9]—were implemented. Results show TA captures leakage features with very few traces, but when traces increase (beyond 7 NTT traces, ≈6,000 butterfly segments), the neural network learns complex non-Gaussian leakage characteristics of real devices, achieving higher accuracy.

## 4.3 Results of Adversarial Training

CNN models were trained for each of the four target operations, with 80% of the profiling traces used for training, and the remaining 20% for validation. As illustrated in Figure 6, the accuracy on both the training and validation sets improves steadily with increasing training epochs. However, when these models were applied to INTT traces, the accuracy (green line) did not show a significant increase. In some cases, it even decreased as training progressed. This observation highlights a significant domain discrepancy between the NTT and INTT operations. The leakage patterns learned from NTT traces are ineffective or even misleading for INTT analysis, necessitating the UDA.

Figure 7 presents the results of adversarial training, where an MLP with three hidden layers served as the discriminator. As adversarial training progressed, the accuracy for INTT samples increased rapidly and converged toward that of NTT samples, achieving only a narrow gap of 1.8% to 5.1% compared to NTT accuracy. This demonstrates the adversarial training structure successfully learned operation-invariant features across NTT and INTT. Notably, the accuracy gap was larger for ADD than for LOAD and STORE, which we attribute to greater variability in ADD operations due to additional operand differences.

In addition to adversarial training, there are also UDA methods based on statistical metrics such as MMD [12] and CMD [35]. We compared these UDA techniques (results in Figure 8) on the ADD traces. Since MMD is sensitive to kernel selection, we evaluated Gaussian, linear, and a complex multi-scale kernel proposed by [13]. Our adversarial approach achieves higher accuracy than other methods; while MMD-based methods can reach comparable performance with complex kernels, our method eliminates sensitive kernel selection and is more automated.

In the adversarial learning framework, the parameter $\lambda$ balances the classification and adversarial losses. We examine its effect on model performance. As shown in Figure 9 (left), high accuracy on INTT is maintained across a wide $\lambda$ interval of $[0.02, 1]$, evaluated over a log-scale range from $10^{-3}$ to $10^{1}$. Excessively large $\lambda$ leads to oversensitivity on domain differences, resulting in uniformly low accuracy on both NTT and INTT (Figure 9, right). Conversely, overly small $\lambda$ leads to slow adaptation, limiting effectiveness within a limited number of iterations.

## 4.4 Results of Key Recovery

Table 2 shows performance on the attack dataset from Subsection 4.1. The profiling-device-free SASCA recovers complete ML-KEM private keys with an average of 6.6 INTT attack traces (100 profiling

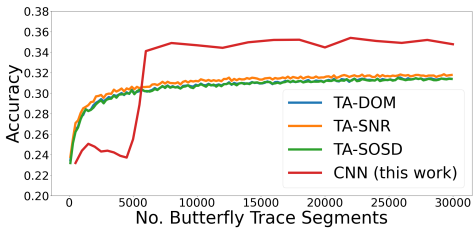
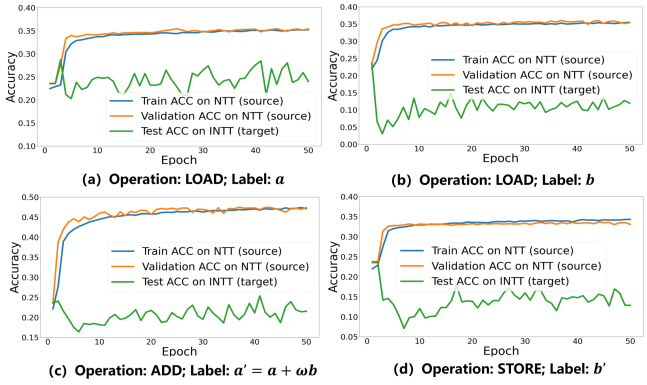



**Figure 5: The Accuracy of CNN and TA.**




(a) Operation: LOAD; Label: $a$ (b) Operation: LOAD; Label: $b$

(c) Operation: ADD; Label: $a' = a + \omega b$ (d) Operation: STORE; Label: $b'$

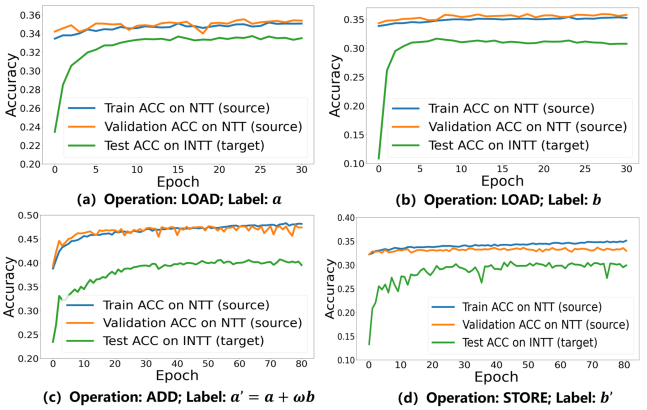
**Figure 6: Before Adversarial Training.**

traces), validating the framework's feasibility. For comparison with existing SASCAs [23], we profiled and attacked the INTT operation directly, needing only 2.6 attack traces (100 profiling traces). Note that this is under a laboratory setting where the profiling and attack devices are the same; if different devices are used for profiling, more attack traces would be required. While the proposed approach uses more attack traces (6.6 vs. 2.6) than profiling-device-dependent SASCA, it eliminates the need for a profiling device. The four additional traces are a reasonable trade-off, and the method still requires relatively few attack traces overall.

A typical target of non-profiling attacks against ML-KEM decapsulation is multiplications in the NTT domain [32]. On the target device of this paper, CPA fails to recover the complete key with 400 attack traces. Compared with traditional non-profiling methods, this work inherits SASCA's ability to comprehensively utilize the leakage of multiple intermediates, thereby significantly reducing the number of required traces.

## 5 Discussion

This section discusses the impact of countermeasures, including shuffling and masking. Owing to their additional overhead, decapsulation prioritizes protecting INTT instances handling secret information, while NTT operations on ciphertext are not primary




(a) Operation: LOAD; Label: $a$ (b) Operation: LOAD; Label: $b$

(c) Operation: ADD; Label: $a' = a + \omega b$ (d) Operation: STORE; Label: $b'$

**Figure 7: After Adversarial Training.**

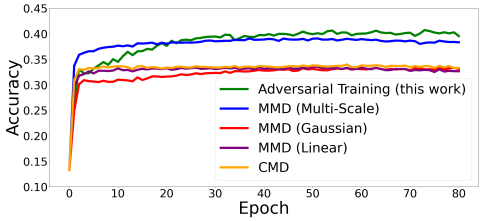



**Figure 8: Comprarison with Other UDA Techniques.**

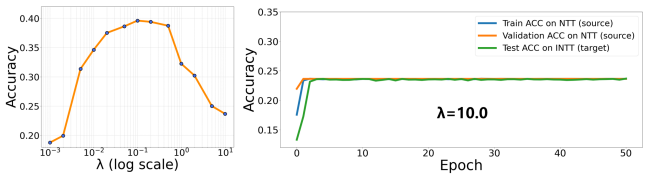



**Figure 9: Effect of the Parameter $\lambda$.**

**Table 2: Attack Performance.**

| Method | Profiling Device | Profiling Traces | Attack Traces | Total |
|---|---|---|---|---|
| this work | × | 100 | 6.6 | 106.6 |
| profiling (SASCA) | ✓ | 100 | 2.6 | 102.6 |
| non-profiling (CPA) | × | - | > 400 | > 400 |

targets of protection. For shuffled INTT, our framework can integrate with Hermelink et al.'s matching algorithm and "shuffle nodes" to deshuffle [17]. For masked implementations, as widely shown in literature [15, 17, 23, 25], SASCA (and our framework) can break masking by attacking the two shares of masked INTT separately, akin to two independent attacks. If NTT is also protected, combining our framework with the end-to-end profiling SCAs [19] is a possible solution, which we leave for future work. Our work highlights that even NTT/INTT on non-sensitive data requires SCA countermeasures.

## 6 Conclusion

This paper presents the first profiling-device-free SASCA framework, better suited to real-world scenarios and opening new possibilities for SASCA targeting ML-KEM. Leveraging the similarity between INTT and NTT operations, we demonstrate the feasibility of profiling the non-sensitive NTT to attack sensitive INTT. Our approach employs adversarial learning to learns operation-invariant side-channel features, enabling cross-operation analysis. Experiments on the Atmel SAM4S microprocessor show that our method recovers the complete key using only 100 profiling traces and an average of 6.6 attack traces.

## References


[1] Guilhèm Assael, Philippe Elbaz-Vincent, and Guillaume Reymond. 2023. improving single-trace attacks on the number-theoretic transform for cortex-M4. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 111–121.

[2] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.

[3] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. 2021. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 27–56.

[4] Pei Cao, Hongyi Zhang, Dawu Gu, Yan Lu, and Yidong Yuan. 2022. AL-PA: cross-device profiled side-channel attack using adversarial learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 691–696.

[5] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. 2002. Template attacks. In *International workshop on cryptographic hardware and embedded systems*. Springer, 13–28.

[6] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

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

[8] W Morven Gentleman and Gordon Sande. 1966. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*. 563–578.

[9] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. 2006. Templates vs. stochastic methods: A performance analysis for side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 15–29.

[10] Anupam Golder, Debayan Das, Josef Danial, Santosh Ghosh, Shreyas Sen, and Arijit Raychowdhury. 2019. Practical approaches toward deep-learning-based cross-device power side-channel attack. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2720–2733.

[11] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[12] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *The journal of machine learning research* 13, 1 (2012), 723–773.

[13] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K Sriperumbudur. 2012. Optimal kernel choice for large-scale two-sample tests. *Advances in neural information processing systems* 25 (2012).

[14] Suvadeep Hajra, Siddhartha Chowdhury, and Debdeep Mukhopadhyay. 2024. Estranet: An efficient shift-invariant transformer network for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 1 (2024), 336–374.

[15] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine Van Vredendaal. 2021. Chosen ciphertext k-trace attacks on masked cca2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 88–113.

[16] Julius Hermelink, Silvan Streit, Erik Mårtensson, and Richard Petri. 2025. A Generic Framework for Side-Channel Attacks Against LWE-Based Cryptosystems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 3–32.

[17] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. 2023. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), 60–88.

[18] Adeline Langlois and Damien Stehlé. 2015. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography* 75, 3 (2015), 565–599.

[19] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. 2021. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR transactions on cryptographic hardware and embedded systems* (2021), 235–274.

[20] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power analysis attacks: Revealing the secrets of smart cards*. Springer.

[21] Catinca Mujdei, Lennert Wouters, Angshuman Karmakar, Arthur Beckers, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. 2024. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Transactions on Embedded Computing Systems* 23, 2 (2024), 1–23.

[22] NIST. 2024. *Module-lattice-based key-encapsulation mechanism standard*. Technical Report. National Institute of Standards and Technology, Gaithersburg, MD.

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

[24] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. 2023. Sok: Deep learning-based physical side-channel analysis. *Comput. Surveys* 55, 11 (2023), 1–35.

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

[26] Thomas Roche. 2025. EUCLEAK Side-Channel Attack on the YubiKey 5 Series (Revealing and Breaking Infineon ECDSA Implementation on the Way). In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4026–4043.

[27] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. 2021. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*. 231–248.

[28] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. 2023. Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, 3 (2023), 473–503.

[29] Benjamin Timon. 2019. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), 107–131.

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

[31] Yuxuan Wang, Jintong Yu, Shipei Qu, Xiaolin Zhang, Xiaowei Li, Chi Zhang, and Dawu Gu. 2025. Mind the Faulty Keccak: A Practical Fault Injection Attack Scheme Apply to All Phases of ML-KEM and ML-DSA. *IEEE Transactions on Information Forensics and Security* (2025).

[32] Yipei Yang, Zongyue Wang, Jing Ye, Junfeng Fan, Shuai Chen, Huawei Li, Xiaowei Li, and Yuan Cao. 2023. Chosen ciphertext correlation power analysis on Kyber. *Integration* 91 (2023), 10–22.

[33] Jonathan S Yedidia, William Freeman, and Yair Weiss. 2000. Generalized belief propagation. *Advances in neural information processing systems* 13 (2000).

[34] Jintong Yu, Yuxuan Wang, Shipei Qu, Yubo Zhao, Yipeng Shi, Pei Cao, Xiangjun Lu, Chi Zhang, Dawu Gu, and Cheng Hong. 2025. End-to-End Non-profiled Side-Channel Analysis on Long Raw Traces. In *European Symposium on Research in Computer Security*. Springer, 526–544.

[35] Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. 2017. Central moment discrepancy (CMD) for domain-invariant representation learning. *arXiv preprint arXiv:1702.08811* (2017).