Release the Power of Rejected Signatures: An Efficient Side-Channel Attack on the ML-DSA Cryptosystem

Zheng Liu, An Wang, Congming Wei, Yaoling Ding, Jingqi Zhang, Annyu Liu, Liehuang Zhu

Abstract—**The module-lattice-based digital signature standard, formerly known as CRYSTALS-DILITHIUM, is a lattice-based post-quantum cryptographic scheme. In August 2024, the National Institute of Standards and Technology officially standardized ML-DSA under FIPS 204. ML-DSA generates one valid signature and multiple rejected signatures during a single signing process. Most side-channel attacks targeting ML-DSA have focused solely on the valid signature, while largely neglecting the hints contained in rejected signatures. Building on prior SASCA frameworks originally proposed for ML-DSA, in this paper we present an efficient and fully practical instantiation of a private-key recovery attack on ML-DSA that jointly exploits side-channel leakages from both valid and rejected signatures within a unified factor graph. This concrete instantiation maximizes the information extracted from a single signing attempt and minimizes the number of required traces for full key recovery.**

We conducted a proof-of-concept experiment with both reference and ASM-optimized implementations on a Cortex-M4 core chip, where the results demonstrate that incorporating rejected signatures reduces the required number of traces by at least 50.0% for full key recovery. Moreover, we show that using only rejected signatures suffices to recover the key with fewer than 30 traces under our setup. Our findings highlight that protecting rejected signatures is crucial, as their leakage provides valuable side-channel information. We strongly recommend implementing countermeasures for rejected signatures during the signing process to mitigate potential threats.

Index Terms—**ML-DSA, Side-Channel Attacks, rejected signature, Post-quantum cryptography.**

I. Introduction

To address the potential threats posed by large-scale quantum computers, NIST initiated the post-quantum cryptography standardization project in 2016. In 2025, this project entered its fourth round, and three digital signature algorithms have been standardized, namely ML-DSA, FN-DSA, and SLH-DSA. Among them, ML-DSA stands out for its excellent overall performance. During the selection process of the project, NIST focuses not only on the performance of the algorithms but also on the minimum overhead required to defend against Side-Channel Attacks (SCA). Therefore, side-channel attacks and defenses on ML-DSA have been a focus of research in recent years.

Zheng Liu, An Wang, Yaoling Ding, Jingqi Zhang, Annyu Liu, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: liuzheng98@bit.edu.cn; wanganl@bit.edu.cn; dyl19@bit.edu.cn; zhangjq@bit.edu.cn; annvliu@bit.edu.cn; liezhuangz@bit.edu.cn). Congming Wei is with the School of Computer and Cyber Sciences, Communication University of China, Beijing, China (e-mail: cmwei@cuc.edu.cn).

Corresponding author: Congming Wei.

SCA extracts secret information stored in cryptographic devices by analyzing side information leaked during their execution. Since Kocher et al. introduced SCA in [1], it has been widely applied to various cryptographic devices. Common types of side information include power traces, Electromagnetic (EM) emissions, and timing variations.

SCAs are generally categorized into profiling attacks and non-profiling attacks. Profiling attacks assume an attacker with access to a modeling device identical to the victim device. Therefore, an attacker can train a model on the modeling device and then apply the model to the victim device. Through this approach, attackers can achieve better performance in their attacks. Typical profiling attacks include the multivariate Gaussian model–based method [2], stochastic models, and deep learning–based side-channel attacks. Non-profiling attacks do not require attackers to have a modeling device in advance. Instead, attackers typically assume a leakage model of the victim device for the attack. As a result, non-profiling attacks generally offer greater flexibility but tend to have lower performance. Typical non-profiling attacks include correlation power analysis and collision attacks.

A. Related Work

Many profiling and non-profiling attacks targeting ML-DSA have been proposed as shown in Table I. In the non-profiling aspect, the first differential power analysis attack against ML-DSA was proposed by Ravi et al. [3], who targeted the multiplication process of the challenge c and private key s_1 and provided simulation results. They also demonstrated that in the ML-DSA algorithm, an attacker does not need to fully recover the private key, but recovering only s_1 is sufficient to forge signatures.

Subsequent researches [4]–[11] have made various improvements based on Ravi et al.'s work. Among them, a notable study by Chen et al. [11] conducted a detailed analysis of the multiplication between c and s_1, as well as the subsequent reduction process, and identified two positions susceptible to the correlation power analysis. Qiao et al. [4] aimed to analyze the Number Theoretic Transform (NTT) result of the private key such as $\hat{s}_1 = \mathrm{NTT}(s_1)$, $\hat{s}_2 = \mathrm{NTT}(s_2)$. They represented the calculation process of $\hat{s}_1 = \mathrm{NTT}(s_1)$ as $As_1 = \hat{s}_1$, transforming the NTT domain recovery problem into a small integer solution problem, which can be solved using the LLL or BKZ algorithm.

Qiao et al. [12], [13] recovered partial bit information of y, constructed equations related to s_1, and solved for s_1 using

TABLE I
SUMMARY OF EXISTING WORKS AND COMPARISON WITH THIS WORK

Work	Attack Target	Attack Type	Experiment Type	Signature Used	Main Method*
[3]	cs_1	Non-Profiled	Simulation	Valid	DPA
[4]–[11]	cs_1	Non-Profiled	Practical	Valid	CPA
[12], [13]	y	Non-Profiled	Practical	Valid	LSM
[16]	w_0	Profiled	Practical	Valid	LSM
[14], [15]	y, cs_1	Profiled	Practical	Valid	ILP
[19]–[21]	s_1, s_2	Profiled	Practical	Valid	DL
[17]	z, c	Profiled	Practical	Rejected	ILP
[18]	y, z, cs_1	Profiled	Simulation	Valid or Rejected +	BP
This Work	y, c, cs_1	Profiled	**Practical**	**Valid and Rejected**	BP

* DPA stands for differential power analysis. CPA stands for correlation power analysis. LSM stands for least squares method. ILP stands for integer linear programming. DL stands for deep learning.
+ [18] presents a generic framework that, in principle, can accommodate both valid and rejected signatures; however, their experimental evaluation considers these sources only in isolation.

the least squares method. [12] and [13] are unique in that they belong to non-profiling attacks but utilize profiling techniques during the attack process.

In the profiling aspect, early attacks focused on directly targeting operations related to s_1 and s_2, such as the unpacking process of s_1 and s_2, or transformations like NTT(s_1) and NTT(s_2). However, in practical deployment scenarios, such computations may only occur once or not at all, making these attack methods difficult to apply in real-world settings.

Subsequent researches generally follow the same approach: leveraging SCA to get hints about the private key, and using different methods to solve for the private key. [14]–[17] obtain numerical hints about s_1 and solve it using the least squares method or integer linear programming. In particular, Bronchain et al. [18] introduce a generic SASCA framework for ML-DSA that can flexibly incorporate various kinds of leakage and side-channel hints (on y, z, rejection events, etc.) into a unified factor graph, and recover s_1 via belief propagation.

B. Motivation

ML-DSA generates signatures through a rejection sampling process, which means that the creation of a valid signature is typically accompanied by several rejected ones. As shown in Table II, the ratios of rejection counts for different security levels of ML-DSA are presented, where the "Expected" column gives the theoretical expectations specified in the algorithm standard [22]. It can be observed that, across all security levels, there is approximately a 75% probability that a signing attempt will produce at least one rejected signature. When the number of signing attempts is three or more, this probability exceeds 98%. Relying solely on valid signatures for key recovery results in a significant waste of information contained in rejected signatures.

There are several works that study the side-channel leakage of rejected signatures in ML-DSA. Bronchain et al. [18] propose a generic SASCA framework for ML-DSA and explicitly remark that both accepted and rejected coefficients could, in principle, be exploited within the same factor graph, but they do not provide an implementation-level evaluation of such combined use. Zhou et al. [17] exploit leakage from the rejection sampling procedure and present a practical key-recovery attack that relies solely on rejected signatures, but their attack still requires about 10^6–10^7 signing attempts across different parameter sets, indicating that the information contained in rejected signatures is not yet efficiently exploited in practice. Motivated by these limitations, this paper provides an implementation-level evaluation of a BP-based attack that jointly exploits valid and rejected signatures on a concrete ML-DSA implementation, and analyzes the practical feasibility and efficiency of such combined use.

TABLE II
STATISTICS ON THE NUMBER OF REJECTED SIGNATURES IN ML-DSA

Rejection	Expected	0	1	2	3	4	5	>5
ML-DSA-44	3.25	0.23	0.18	0.14	0.11	0.08	0.06	0.20
ML-DSA-65	4.10	0.19	0.16	0.12	0.10	0.08	0.07	0.28
ML-DSA-87	3.85	0.25	0.19	0.14	0.11	0.08	0.06	0.17

C. Contribution

The contributions of this paper are as follows.

- Building on the generic SASCA framework for ML-DSA [18], this paper presents a practical and efficient BP-based private-key recovery attack on ML-DSA that makes use of the side-channel leakages generated during a single signing attempt, including those from both valid and rejected signatures. To achieve this, we instantiate an ML-DSA-specific factor-graph construction that accommodates leakages on INTT(cs_1), NTT(c), and rejection events within a single BP instance, thereby maximizing the utilization of all available hints generated during a single signing attempt. Besides, we consider a more general and realistic attack scenario that encompasses more commonly targeted operations, lower signal-to-noise ratio (SNR) in optimized implementations, and more relaxed modeling requirements.
- We systematically investigate the leakages of INTT(cs_1) and NTT(c) for both valid and rejected signatures, and instantiate a concrete BP-based attack that exploits these leakages to make practical use of rejected signatures. We

also investigate the impact of the accuracy of the challenge c associated with rejected signatures on the attack performance, a factor that has been ignored in previous studies [17], [18]. Furthermore, while the possibility of using soft information for c has been mentioned at a high level in prior work [18], we instantiate and tailor a soft-information-based BP message-update rule for c in the concrete ML-DSA setting, and we experimentally demonstrate that this significantly mitigates the degradation in attack performance caused by insufficient accuracy of c.

- We perform an implementation-level evaluation of the proposed attack on both the reference implementation and an ASM-optimized implementation of ML-DSA on a Cortex-M4 platform. The experimental results show that using all signatures reduces the number of traces required for key recovery by at least 50% compared to using only valid signatures. When using only the rejected signatures, fewer than 30 traces are sufficient to recover the key, which is significantly fewer than in previous study [17]. Our experiments on varying the accuracy of c show that even small fluctuations in the accuracy of c can significantly affect the attack performance, especially in highly optimized implementations. In addition, our strategy of using the soft information of c enables us to achieve the attack performance that would otherwise require a much higher c accuracy.

All code and datasets used in this paper are publicly available at https://github.com/AIGIUS/BP-On-ML-DSA.

II. Background

A. Notations

We adopt the notations from [23] and introduce several custom symbols specific to this work, as listed below. It should be noted that the symbol a in the following context does not refer to any specific variable; it is used solely to illustrate different forms of variable representation.

$\mathbb{Z}$: the ring of integers.
$\mathbb{Z}_q$: the ring of polynomials modulo q.
$\mathbb{R}_q$: the quotient ring of polynomials modulo X^n+1, also written as $\mathbb{Z}_q[X]/(X^n+1)$.
$[i,j]$: the set $\{m \mid i \le m \le j, m \in \mathbb{Z}\}$.
$\mathbf{a}$: boldface is used to denote vectors or matrices.
a : non-boldface symbols denote a polynomial or an integer.
$\mathbf{a}_i$: the i-th element of a vector $\mathbf{a}$.
a_i : the i-th coefficient of a polynomial a.
$a_{[i:j]}$: the bits from position i to j of an integer a.
$\hat{a}$: the NTT representation of polynomial a, $\hat{a} = \mathrm{NTT}(a)$.
$*$: the multiplication of two polynomials, typically refers to convolution multiplication.
$\bar{\ }$: $\bar{z}$,$\bar{c}$ denote the rejected signatures in ML-DSA, while z,c represent valid signatures.
S_η : the set of polynomials whose coefficients lie in the range $[-\eta,\eta]$.
B_τ : the set of polynomials in which exactly τ coefficients are equal to ±1, and all remaining coefficients are zero.

B. ML-DSA

ML-DSA is one of the three digital signature algorithms standardized by NIST. ML-DSA supports three security levels, namely ML-DSA-44, ML-DSA-65, and ML-DSA-87. Its security is primarily based on the lattice-based module learning with errors problem and the module short integer solution problem. ML-DSA consists of three parts: key generation, signing, and verification. In this paper, we mainly explain the key generation and signing parts. The verification part is generally not targeted for SCA, so it is not covered. For the sake of brevity, we present only the algorithm templates and the necessary details, as shown in Algorithm 1 and Algorithm 2.

Algorithm 1 ML-DSA key generation algorithm

```
1: $\mathbf{A} \leftarrow \mathbb{R}_q^{k\times l}$
2: $(\mathbf{s}_1,\mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$
3: $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
4: return $(pk = (\mathbf{A},\mathbf{t}), sk = (\mathbf{A},\mathbf{t},\mathbf{s}_1,\mathbf{s}_2))$
```

Algorithm 2 ML-DSA signing algorithm

```
1: $\mathbf{z} = \perp$
2: while $\mathbf{z} = \perp$ do
3:     $\mathbf{y} \leftarrow S_{\gamma_1-1}^l$
4:     $\mathbf{w} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \cdot \mathrm{NTT}(\mathbf{y}))$
5:     $\mathbf{w}_1 = \mathrm{HighBits}(\mathbf{w}, 2\gamma_2)$
6:     $c \in B_\tau = \mathrm{H}(M||\mathbf{w}_1)$
7:     $\mathbf{z} = \mathbf{y} + \mathrm{NTT}^{-1}(\mathrm{NTT}(c)\cdot\hat{\mathbf{s}}_1)$
8:     if $||z||_\infty \ge \gamma_1 - \beta$ or $||\mathrm{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)||_\infty \ge \gamma_2 - \beta$ then
9:         $\mathbf{z} = \perp$
10:    end if
11: end while
12: return $\sigma = (\mathbf{z}, c)$
```

C. Template Attacks

In this paper, we adopt the multivariate Gaussian model–based method proposed in [2]. For clarity and conciseness, the term the template attack hereafter refers specifically to this method. The template attack consists of two phases: the modeling phase and the attack phase. In the modeling phase, the modeling device runs n times, recording the input $\mathbf{p}_i$, the corresponding output σ_i, and the associated power trace $\mathbf{t}_i$, where $i \in [0, n-1]$. Using the key, inputs, and corresponding output, the intermediate values $\mathbf{v}_i$ that the attacker aims to attack can be computed. The power traces are then categorized based on $\mathbf{v}_i$.

Assuming $\mathbf{v}_i$ can be divided into d different categories. Let the power trace belonging to the j-th category be denoted as $\mathbf{t}_{j,i}$, where $i \in [0, \mathbf{d}_j - 1]$, $\mathbf{d}_j$ represents the number of traces in the j-th category. The mean vector $\mathbf{m}_j$ and the covariance matrix $\mathbf{C}_j$ for the j-th category can be calculated using the following formulas:

$$\mathbf{m}_j = \frac{1}{\mathbf{d}_j}\sum_{i=0}^{\mathbf{d}_j-1}\mathbf{t}_{j,i},$$

$$\mathbf{C}_j = \frac{1}{\mathbf{d}_j}\sum_{i=0}^{\mathbf{d}_j-1}(\mathbf{t}_{j,i}-\mathbf{m}_j)^T(\mathbf{t}_{j,i}-\mathbf{m}_j).$$

In the attacking phase, the attacker collects $n^{'}$ power traces $\mathbf{t}_i^{'}$ when running the victim device, where $i \in [0, n^{'}-1]$. The probability that $\mathbf{t}_i^{'}$ belongs to the j-th template can be calculated using the following formula:

$$P_{i,j}(\mathbf{t}_i^{'};(\mathbf{m}_j,\mathbf{C}_j)) = \frac{exp(-\frac{1}{2}(\mathbf{t}_i^{'}-\mathbf{m}_j)\mathbf{C}_j^{-1}(\mathbf{t}_i^{'}-\mathbf{m}_j)^T)}{\sqrt{(2\pi)^r det(\mathbf{C}_j)}},$$

where r is the dimension of $\mathbf{t}_i$ and $\mathbf{t}_i^{'}$.

D. Number Theoretic Transform

Owing to its high computational efficiency, the NTT is widely used in lattice-based cryptography and related cryptographic applications [24].

Given a polynomial $a \in \mathbb{R}_q$, its NTT transformation is defined as $\hat{a} = \text{NTT}(a)$, as shown in Equation 1, where ω is the primitive n-th root of unity modulo q.

$$\hat{a}_j = \sum_{i=0}^{n-1}\omega^{ij}a_i \ mod\ q \tag{1}$$

The inverse NTT (INTT) is similarly defined as $a = \text{INTT}(\hat{a})$, as shown in Equation 2, where n^{-1} is called the scaling factor.

$$a_i = n^{-1}\sum_{j=0}^{n-1}\omega^{-ij}\hat{a}_j \ mod\ q \tag{2}$$

In practice, the Cooley-Tukey (CT) algorithm is commonly used for efficient computation of the NTT. The fundamental computational unit in the CT algorithm is the CT butterfly unit. As shown in Figure 1, a CT butterfly unit performs the computation of $C = A + B \times \omega^k$ and $D = A - B \times \omega^k$. By configuring multiple such units in a structured manner, a complete NTT computation can be efficiently executed.

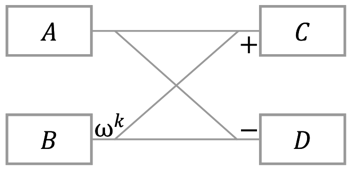


Fig. 1. The Cooley-Tukey butterfly unit. The complete NTT process is composed by stacking multiple such units together.

E. Belief Propagation

BP is a probabilistic inference algorithm widely used in graphical models, particularly in factor graphs, to compute marginal distributions of variables. A factor graph consists of factor nodes and variable nodes. Variable nodes represent the variables, while factor nodes represent the constraints imposed on these variables. BP operates by iteratively passing messages between variable nodes and factor nodes in a factor graph. These messages represent probability distributions, allowing the algorithm to update its belief about each variable until convergence is achieved.

The message sent from a factor node a to a variable node i is computed as the product of all incoming messages from a's neighboring nodes (excluding i), multiplied by the factor function of a, and then marginalized over i. This is formally expressed in the following formula:

$$m_{a\to i}(x_i) = \sum_{x_{N(a)\setminus\{i\}}} f_a(x_i, x_{N(a)\setminus\{i\}}) \prod_{j\in N(a)\setminus\{i\}} m_{j\to a}(x_j),$$

where $N(a)$ denotes the set of neighboring nodes of a.

The message sent from a variable node i to a factor node a is computed as the product of all incoming messages from i's neighboring nodes (excluding a). This process is illustrated in the following formula:

$$m_{i\to a}(x_i) = \prod_{b\in N(i)\setminus\{a\}} m_{b\to i}(x_i).$$

BP was introduced into SCA in [25]. In this context, the results of SCA are typically referred to as hints. These hints are used to initialize the factor nodes in the factor graph, after which the BP algorithm is executed to solve for the private key.

III. General Attack Framework

The core idea of this paper is to maximize the utilization of all side-channel leakages within a single signing attempt to recover the private key as efficiently as possible. For each ML-DSA signing attempt, we can capture a corresponding side-channel trace. Each trace contains leakages from one valid signature and multiple rejected signatures, as illustrated in Figure 2. Assuming that four signatures are sufficient to recover the private key, an attacker using only valid signatures would require four traces for key recovery. However, by leveraging both valid and rejected signatures, the attacker has the opportunity to recover the private key using fewer traces, as illustrated in Figure 2.

A. Overview of the Attack Strategy

This section outlines the overall attack framework proposed in this paper. The framework consists of four main processes, as illustrated in Figure 3. Typically, profiled SCA consists of two phases, which are the modeling phase and the attack phase. For the sake of clarity, we only present the attack phase, omitting the modeling phase. We describe the four stages as follows:

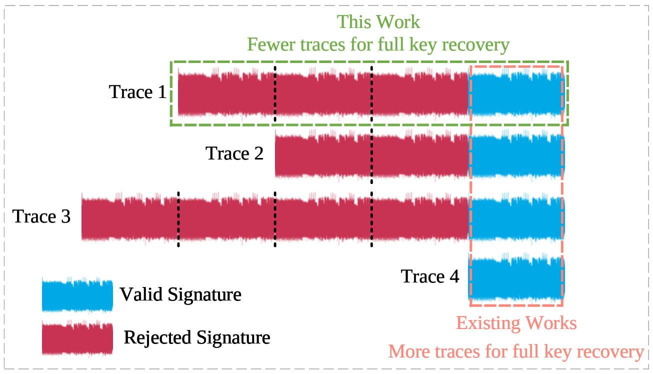


Fig. 2. Main idea of the attack. By simultaneously leveraging both valid and rejected signatures (which occur with high probability), our method offers the potential to recover the private key using fewer traces.

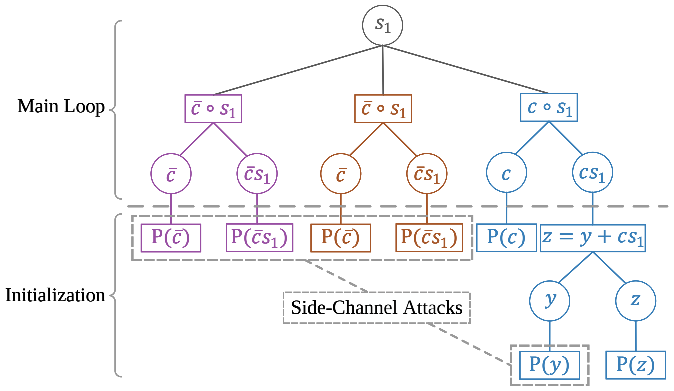


Fig. 4. The factor graph used in this paper. $\bar{c}$ denotes rejected signature and c denotes valid signature. Different colors represent different signature instances. $P(\bar{c})$, $P(\bar{c}s_1)$, and $P(y)$ represent hints obtained through SCA.

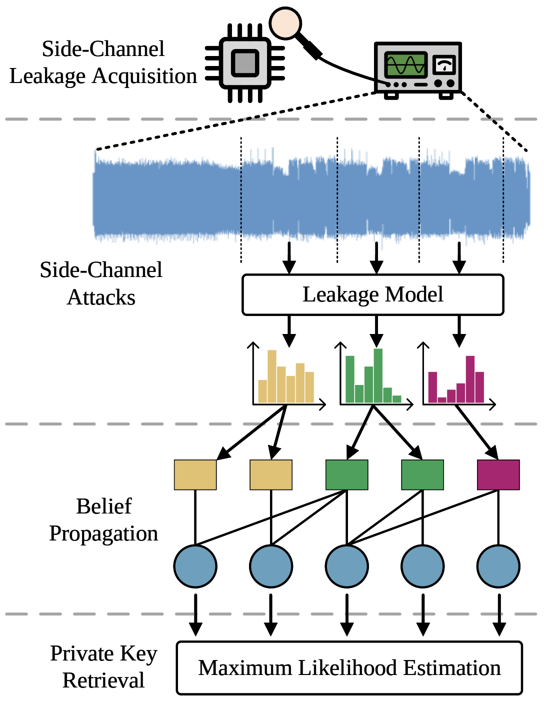


Fig. 3. General attack framework. For the sake of simplicity, we omit the modeling phase.

- **Side-Channel Leakage Acquisition**: The attacker collects side-channel information from the victim's device. In this work, we collect EM traces near the voltage regulator pin of the victim device to achieve a higher signal-to-noise ratio.
- **Side-Channel Attacks**: The attacker identifies the positions of rejected signatures and valid signatures within the traces and classifies them using the leakage model obtained from the modeling device. The classification process yields hints of the target variables. Specifically, these variables are as follows:
  - Valid Signatures: y, cs_1
  - Rejected Signatures: $\bar{c}$, $\bar{c}s_1$
- **Belief Propagation**: The attacker constructs a factor graph, initializing it with hints obtained from the leakage model. The BP algorithm is then executed iteratively until the messages in the factor graph converge.
- **Private Key Retrieval**: Once the BP algorithm converges, the attacker can recover the private key from variable nodes. A common method for this is maximum likelihood estimation.

The following subsections will discuss specific details and issues related to the general attack framework.

B. Choices in Side-Channel Attacks

Common techniques for constructing leakage models include the template attack, stochastic model, and deep learning. In this work, we employ the template attack. While deep learning may potentially yield more accurate leakage models, the template attack is chosen for its advantages, including parameter-free implementation and consistently reliable performance. This makes it well-suited for proof-of-concept validation.

This work targets $\bar{c}s_1$ instead of $\bar{z}$, for existing research has shown that the hint from $\bar{z}$ is too limited [17], [18]. Incorporating $\bar{z}$ might introduce additional noise, degrading performance. In addition, SCA on $\bar{z}$ is not the primary focus of this paper. Nevertheless, we demonstrate that $\bar{z}$ can be easily integrated into the proposed attack framework in Section III-C. The details of the SCA on y, cs_1 (or $\bar{c}s_1$), and $\bar{c}$ will be described in Section IV.

C. Factor Graph Construction

As shown in Figure 4, the factor graph used in this paper is illustrated. Since c and z (as part of the valid signature) are known to the attacker, they could be integrated into the factor node $z = y + cs_1$. However, we still represent $P(c)$ and $P(z)$ as independent factor nodes in the factor graph, where only one value has a probability of 1, and all others have a probability of 0. This enhances the generality of the proposed factor graph. If an attacker can obtain a sufficiently accurate hint about the rejected z, it can be incorporated into our factor graph simply by replacing z with $\bar{z}$.

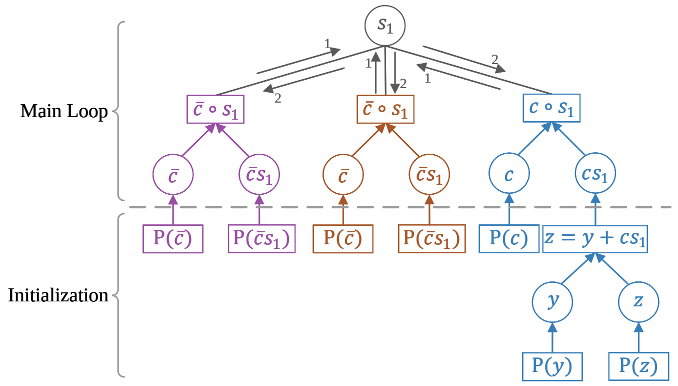


Fig. 5. Message passing path in the factor graph. Only the edges directly connected to the private key participate in bidirectional message propagation, while all other edges perform upward propagation only.

The factor graph includes two types of constraints in factor nodes: polynomial multiplication ($\bar{c} * s_1$, $c * s_1$) and polynomial addition ($z = y + cs_1$). Constraints $c * s_1$ and $\bar{c} * s_1$ follow the Equation 3, where $\zeta = 0$ when $j \leq i$, $\zeta = 1$ when $j > i$, $i \in [0, 255]$. Constraint $z = y + cs_1$ follows the Equation 4, where $i \in [0, 255]$.

$$(cs_1)_i = \sum_{j=0}^{255} (-1)^{\zeta} c_{(i-j) \ mod \ 256} \times (s_1)_j \tag{3}$$

$$z_i = y_i + (cs_1)_i \tag{4}$$

Compared to the factor graph in existing work [18], the factor graph proposed in this paper represents c (as well as $\bar{c}$) and z as probability distributions, enabling the utilization of both valid and rejected signatures. Additionally, the factor graph in this paper explicitly represents the constraint $z = y + cs_1$, allowing s_1 to be directly recovered from SCA results. In contrast, existing work needs to derive the distribution of cs_1 from y before incorporating it into the factor graph. Our work is more automated, reducing manual processing steps.

D. *Efficient BP computation*

In the factor graph used in this paper, the BP computational overhead mainly comes from three sources: the iterative message passing process, the message size, and the computation process of the constraints. The following content will discuss these three parts separately.

In a standard BP algorithm, each iteration requires bidirectional message passing along every edge in the factor graph. However, in the factor graph proposed in this paper, this process can be pruned to reduce computational complexity. This paper adopts the message passing path as shown in Figure 5.

As shown in Figure 5, the initialization part only consists of values obtained from SCA results or known values. Consequently, messages in this part remain unchanged throughout the BP process, propagating only upward and requiring computation only once during the first iteration of BP.

In the main loop part, only the edges directly connected to s_1 require bidirectional message passing, while all other edges propagate messages upward only. At a higher level, the factor graph of the main loop forms a tree-like structure, thus, the edges directly connected to s_1 do not require parallel bidirectional updates. Instead, the message passing follows a top-down and bottom-up iterative process [26].

Here, we describe the update process of messages 1, and how the soft information of c, namely $\text{P}(c)$, is incorporated into the factor graph. From Equation 3, we can directly get

$$\text{P}((cs_1)_i) = \text{P}(\sum_{j \in [0,255]} (-1)^{\zeta} c_{(i-j) \ mod \ 256} \times (s_1)_j).$$

For clarity, we omit the subscript i and the term $(-1)^{\zeta}$, and derive

$$\begin{aligned} \text{P}(cs_1) &= \text{P}(\sum_{j \in [0,255]} (s_1)_j \times c_j) \\ &= \text{P}((s_1)_k \times c_k) * \text{P}(\sum_{j \in [0,255] \setminus \{k\}} (s_1)_j \times c_j), \end{aligned}$$

where $*$ denotes the discrete convolution, $(s_1)_k$ denotes the k-th coefficient of s_1, and c_k denotes the coefficient that multiplies $(s_1)_k$, which is not necessarily the k-th coefficient of c. Next, we can derive

$$\text{P}((s_1)_k \times c_k) = \text{P}(cs_1) * \text{P}(\sum_{j \in [0,255] \setminus \{k\}} -(s_1)_j \times c_j).$$

The intuition behind this formulation is to estimate, from the perspective of all other coefficients of s_1, the distribution of the product $(s_1)_k \times c_k$.

For clarity, we define

$$\text{R}_k = \mathop{*}_{j \in [0,255] \setminus \{k\}} \text{P}\big(-(s_1)_j \times c_j \big),$$

then we have

$$\text{P}((s_1)_k \times c_k) = \text{P}(cs_1) * \text{R}_k,$$

where $\text{P}((s_1)_j \times c_j)$ can be explicitly derived from $\text{P}((s_1)_j)$ and $\text{P}(c_j)$ by a direct application of the law of total probability since s_1 and c both take values in small finite sets.

We first use $\text{P}((s_1)_j)$ from message 2 and $\text{P}(c_j)$ to compute R_k, and then combine R_k with $\text{P}(cs_1)$ to obtain $\text{P}((s_1)_k \times c_k)$. Finally, we use $\text{P}((s_1)_k \times c_k)$ together with $\text{P}(c_k)$ to update $\text{P}(s_1)$ in message 1. One can expand $\text{P}((s_1)_k \times c_k)$, substitute the possible values of s_1 and c, and solve the resulting system of equations. Since the value domains of both s_1 and c are finite and symmetric, the system is solvable. However, such a procedure is computationally expensive. Therefore, in this work, we adopt a more intuitive approach:

$$\begin{aligned} \text{P}((s_1)_k) = \text{P}(c_k = -1) \times \text{P}(-(s_1)_k \times c_k) \\ + \text{P}(c_k = 1) \times \text{P}((s_1)_k \times c_k), \end{aligned}$$

that is, taking the weighted average of $\text{P}((s_1)_k \times c_k)$ using $\text{P}(c_k)$ when $c = -1$ or $+1$. When $c = 0$, we assume it has no influence on the distribution of $(s_1)_k$. Experimental results demonstrate that this approach can also effectively recover the secret key.

For the update of message 2, it follows directly from the standard definition of message updating for variable nodes in the Section II-E, and is therefore not elaborated here.

It is worth noting that although the messages of the P($\bar{c}$) node are currently propagated in a single direction, allowing bidirectional message propagation at this node could potentially enable the simultaneous recovery of both s_1 and $\bar{c}$. This represents a promising direction for future research, inspired by the reviewer's valuable suggestion.

Considering message size, the main loop contains three types of messages: s_1, c (or $\bar{c}$), and cs_1 (or $\bar{c}s_1$). Since these variables have small value ranges, their impact on BP computational complexity is minimal. In contrast, in the initialization phase, y has a much larger value range of 2^{17} (or 2^{19} for ML-DSA-65 and ML-DSA-87). This results in a larger message size, which not only increases memory overhead but also prolongs constraint computations in factor nodes.

To control the message size of y, we analyze the $z = y + cs_1$ process and limit the bits used for y, thereby reducing computational complexity. The details of this optimization are provided in Section IV-A.

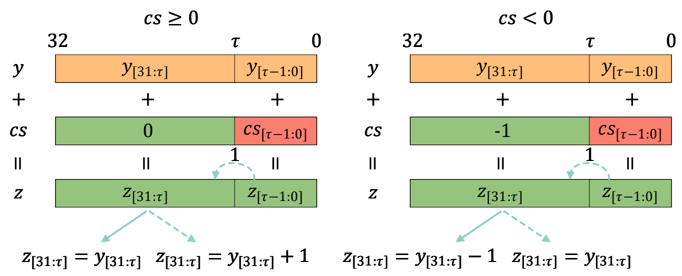
Regarding the computational cost of the constraints, Equation 3 and Equation 4 involve two fundamental distribution operations: the sum of two distributions P$(A + B)$ and the product of two distributions P$(A \times B)$. Since P$(A + B) =$ P$(A) *$ P(B) when A and B are independent, the computation can be accelerated using the NTT. Additionally, since the bits used for y have been optimized, Equation 4 can also be further refined. The details are provided in Section IV-A. For P$(A\times B)$, since the coefficients of c are restricted to $\{-1, 0, 1\}$, it equals P(B), P$(-B)$, or 0, which can be obtained by simply flipping the distribution of B.

IV. Side-Channel Attacks on ML-DSA

This section will describe how to perform SCA on the target variables y, cs_1 (or $\bar{c}s_1$), and $\bar{c}$, respectively. For y, we primarily limit the number of bits recovered through SCA to reduce the message size in the initialization part of the factor graph, thereby lowering the computational overhead of BP. For $\bar{c}$ and cs_1 (or $\bar{c}s_1$), we describe their leakage and the corresponding SCA, ensuring the effective utilization of rejected signatures.

A. Side-Channel Attack on y

In this section, we present how to perform SCA on y. We directly recover the numerical value of y by attacking its generation process, which corresponds to line 3 of Algorithm 2. Since each coefficient of y is bounded by γ_1 (2^{17} for ML-DSA-44 and 2^{19} for ML-DSA-65 and ML-DSA-87), modeling all bits of y's coefficients not only complicates the modeling process but also significantly increases the overhead of message passing in BP and the computation cost of the constraints. Therefore, minimizing the number of bits of y to be modeled is a primary consideration. Although some studies have discussed this issue [12]–[15], the question of "what is the minimum number of bits of y that must be modeled to accurately recover the distribution of cs_1" has not yet been fully answered. To address this issue, we propose and prove the following proposition in this section.



cs ≥ 0
32
τ
0
y
$y_{[31:\tau]}$
$y_{[\tau-1:0]}$
+
cs
0
$cs_{[\tau-1:0]}$
1
z
$z_{[31:\tau]}$
$z_{[\tau-1:0]}$
$z_{[31:\tau]} = y_{[31:\tau]}$
$z_{[31:\tau]} = y_{[31:\tau]} + 1$
cs < 0
-1
$z_{[31:\tau]} = y_{[31:\tau]} - 1$
$z_{[31:\tau]} = y_{[31:\tau]}$

Fig. 6. Relationship between $z_{[31:\tau]}$ and $y_{[31:\tau]}$ determined by the sign of cs_1 and the presence of a carry in $y_{[\tau-1:0]} + (cs_1)_{[\tau-1:0]}$. Dashed lines represent cases with a carry, and solid lines represent cases without.

Proposition 1. *Given $|cs_1| < 2^\tau$, at least $\tau + 2$ bits of y are required to accurately reconstruct the distribution of cs_1, where $z = y + cs_1$, and z, y, c, s_1 refer to the coefficients of the corresponding polynomial.*

Proof. On a typical 32-bit processor, z, y, and cs_1 are stored and computed in the form of 32-bit two's complement integers. We can express y, z, and cs_1 as the sum of their higher $32-\tau$ bits and lower τ bits, as shown in Equation 5.

$$\begin{aligned} z_{[31:\tau]} \times 2^\tau + z_{[\tau-1:0]} = y_{[31:\tau]} \times 2^\tau + y_{[\tau-1:0]} \\ +(cs_1)_{[31:\tau]} \times 2^\tau + (cs_1)_{[\tau-1:0]} \end{aligned} \tag{5}$$

As illustrated in Figure 6, based on the sign of cs_1 and the presence or absence of a carry in the computation of $y_{[\tau-1:0]} + (cs_1)_{[\tau-1:0]}$, four relationships between $z_{[31:\tau]}$ and $y_{[31:\tau]}$ can be derived.

In conjunction with Equation 5, $(cs_1)_{[31:0]}$ can be derived from $z_{[\tau-1:0]}$ and $y_{[\tau-1:0]}$ in all four cases.

1) With carry, $cs \geq 0$:

$$(cs_1)_{[31:0]} = z_{[\tau-1:0]} - y_{[\tau-1:0]} + 2^\tau$$

2) Without carry, $cs \geq 0$:

$$(cs_1)_{[31:0]} = z_{[\tau-1:0]} - y_{[\tau-1:0]}$$

3) With carry, $cs < 0$:

$$(cs_1)_{[31:0]} = z_{[\tau-1:0]} - y_{[\tau-1:0]}$$

4) Without carry, $cs < 0$:

$$(cs_1)_{[31:0]} = z_{[\tau-1:0]} - y_{[\tau-1:0]} - 2^\tau$$

Summarizing the four possible cases, it can be observed that $(cs_1)_{[31:0]}$ depends only on $z_{[\tau-1:0]} - y_{[\tau-1:0]}$ and the relationship between $z_{[31:\tau]}$ and $y_{[31:\tau]}$. Since the relationship between $z_{[31:\tau]}$ and $y_{[31:\tau]}$ only has three distinct cases, recovering the lower 2 bits of $y_{[31:\tau]}$ is sufficient to capture this relationship. Therefore, recovering the lower $\tau + 2$ bits of each y is sufficient to reconstruct the distribution of cs_1 accurately. □

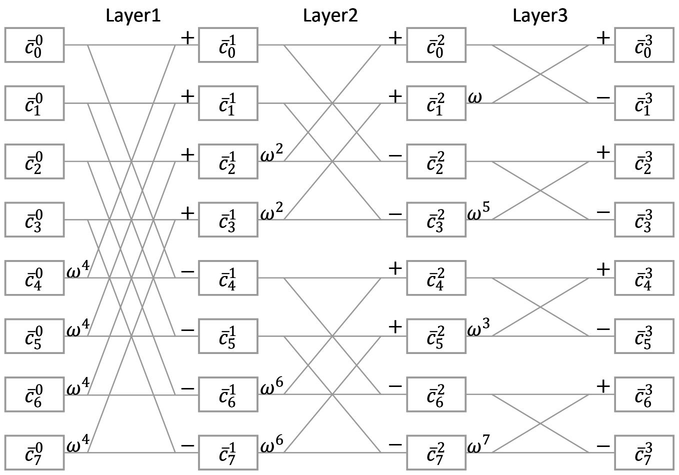

Layer1
Layer2
Layer3
$\bar{c}^0_0$ $\bar{c}^0_1$ $\bar{c}^0_2$ $\bar{c}^0_3$ $\bar{c}^0_4$ $\bar{c}^0_5$ $\bar{c}^0_6$ $\bar{c}^0_7$
ω^4 ω^4 ω^4 ω^4
$\bar{c}^1_0$ $\bar{c}^1_1$ $\bar{c}^1_2$ $\bar{c}^1_3$ $\bar{c}^1_4$ $\bar{c}^1_5$ $\bar{c}^1_6$ $\bar{c}^1_7$
ω^2 ω^2 ω^6 ω^6
$\bar{c}^2_0$ $\bar{c}^2_1$ $\bar{c}^2_2$ $\bar{c}^2_3$ $\bar{c}^2_4$ $\bar{c}^2_5$ $\bar{c}^2_6$ $\bar{c}^2_7$
ω ω^5 ω^3 ω^7
$\bar{c}^3_0$ $\bar{c}^3_1$ $\bar{c}^3_2$ $\bar{c}^3_3$ $\bar{c}^3_4$ $\bar{c}^3_5$ $\bar{c}^3_6$ $\bar{c}^3_7$

Fig. 7. Simplified NTT($\bar{c}$) process. We assume that $\bar{c}$ is a polynomial with only 8 coefficients.

B. *Side-Channel Attack on cs_1*

In this section, we present how to perform SCA on cs_1 (as well as $\bar{c}s_1$). We focus on the final step of INTT($\hat{c}\hat{s}_1$), which corresponds to the multiplication by n^{-1} in Equation 2. Since this operation involves a multiplication combined with Montgomery reduction, it naturally exhibits a high signal-to-noise ratio in SCA. We directly model the lower 7 bits of the coefficients of cs_1, for $||cs_1||_\infty$ is less than 2^6 with nearly 100% probability according to a previous study [14].

C. *Side-Channel Attack on $\bar{c}$*

In this section, we present how to perform SCA to recover the rejected signature $\bar{c}$. We recover $\bar{c}$ by analyzing the process of NTT($\bar{c}$). SCA on the NTT process has been extensively discussed in several studies [17], [27], [28]. Similar to [17], in this work, we apply a layer-by-layer recovery method that neither relies on BP nor requires a full analysis of the NTT process.

Rather than analyzing the entire NTT process, we focus on attacking the first few layers of NTT. First, we analyze the first layer of the NTT. A simplified illustration of the NTT($\bar{c}$) process is shown in Figure 7. Since $\bar{c}^0_i \in \{-1, 0, 1\}$, there are only 9 possible input combinations for each butterfly unit in the first layer. Therefore, 9 templates are required for the attack. Considering each butterfly unit in the figure, the values of $\{\bar{c}^0_i | i \in [4, 7]\}$ significantly influence the Hamming weight (HW) of the butterfly unit's output. Specifically, when $\bar{c}^0_i$ takes values in $\{-1, 0, 1\}$, the Hamming weight of the output corresponds to HW($\bar{c}^0_{i-4} - \omega^4$), HW($\bar{c}^0_{i-4}$), or HW($\bar{c}^0_{i-4} + \omega^4$) separately. Through similar analysis, it can be observed that the values of $\{\bar{c}^0_i | i \in [0, 3]\}$ have a smaller impact on the Hamming weight of the butterfly unit's output. Consequently, the recovery accuracy for $\{\bar{c}^0_i | i \in [4, 7]\}$ is high, while $\{\bar{c}^0_i | i \in [0, 3]\}$ may contain errors.

To correct these errors, the attacker can proceed to attack the second layer. Leveraging the fact that $\{\bar{c}^0_i | i \in [4, 7]\}$ has already been recovered, this constrains $\bar{c}^1_i$ to only three possible values. Similar to the first layer, in each butterfly unit, the second input value is easier to recover, while the first input value is more error-prone. By iteratively attacking subsequent layers, the attacker can progressively refine and correct errors from previous layers, ultimately improving the overall attack accuracy.

V. Experiment Results

A. *Experimental Setup*

The experimental setup is shown in Figure 10. We deployed both the reference and ASM-optimized implementations of ML-DSA-44 on an STM32F4DISCOVERY development board. EM traces were collected using a Langer LF-U 2.5 probe, amplified by a Langer PA 303 amplifier, and captured by a PicoScope 3203D oscilloscope at a sampling rate of 500 MS/s. The target microcontroller was clocked at 168 MHz. In the case where no rejected signatures occur, a complete trace covering one signing attempt contains approximately 18 million samples for the reference implementation and approximately 13 million samples for the ASM-optimized implementation (both compiled with `-O3`). The traces were then transferred to a laptop equipped with an Intel Core i9-14900HX processor and 64 GB RAM for further analysis. We conducted a proof-of-concept experiment on both implementations under the O3 optimization level.

B. *Definition of a Successful Key Recovery*

Before conducting the experiments, we first need to define what constitutes a successful attack. The ultimate objective of attacking ML-DSA is to forge valid signatures. As demonstrated in prior studies, signature forgery requires the recovery of only a portion of the secret key, specifically s_1 [3]. Consequently, in the context of our work, a successful key recovery is defined as the accurate recovery of the value of s_1 which is composed of multiple components. Since these components are processed sequentially in the signing algorithm and exhibit no significant differences in their recovery behavior, the number of traces required to recover each component can be considered approximately the same. Hence, in our experiments, we define a successful key recovery as the complete recovery of the first component of s_1, which is equivalent to the full recovery of s_1 and thus represents a successful key recovery event.

C. *Side-Channel Attacks Results*

Figure 8 shows the EM traces of the reference implementation of ML-DSA on STM32F407 with optimization O3. Figure 8(a) presents the complete trace of a single signing process, which includes two rejected signatures and one valid signature. Figure 8(b) shows a zoomed-in view of the single signature generation process from Figure 8(a). For ease of description, we do not distinguish between c and $\bar{c}$, nor between cs_1 and $\bar{c}s_1$ here. We have marked the main operations during a single signature generation in the Figure 8(b), corresponding to lines 3 to 7 of Algorithm 2. In this trace, the generation of $\mathbf{y}$, the NTT(c) process, and the INTT($c\mathbf{s}_1$) process are the primary targets. As clearly shown in Figure 8(b), both the $\mathbf{y}$ generation

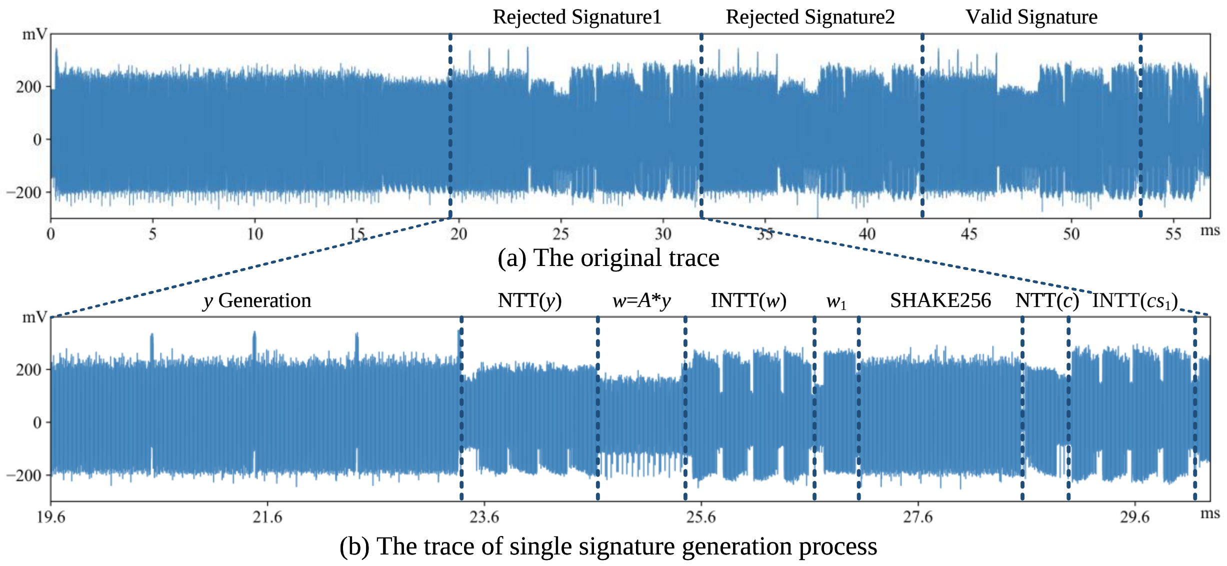


Fig. 8. EM traces of the reference implementation of ML-DSA on STM32F407 with optimization O3.

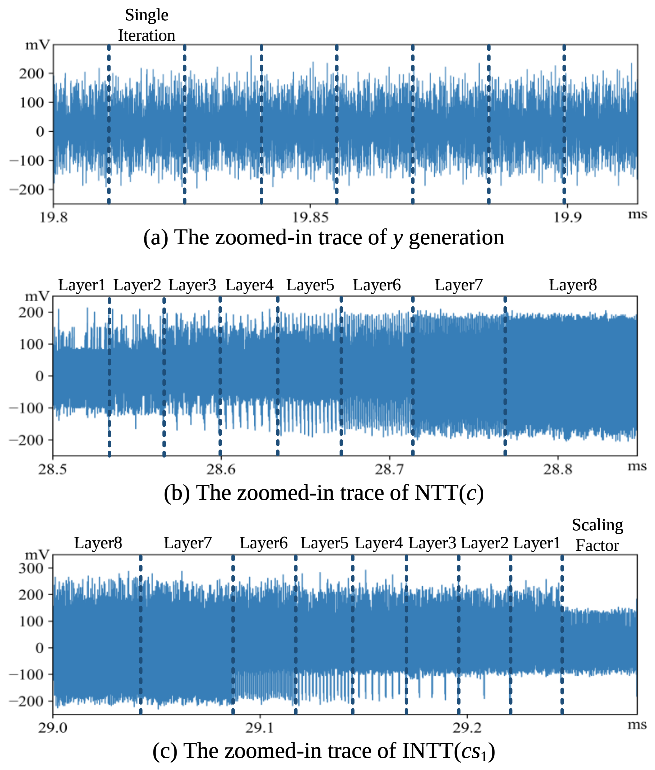


Fig. 9. Zoomed-in traces of y generation, NTT(c), and INTT(cs_1).

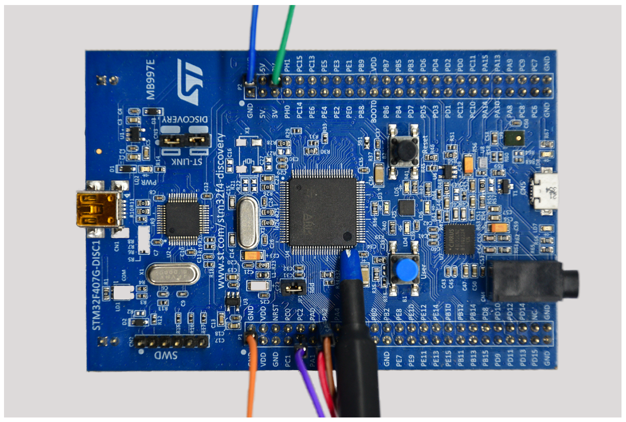


Fig. 10. The experimental setup used in this work. The EM probe is positioned near the VCAP pin of the target chip.

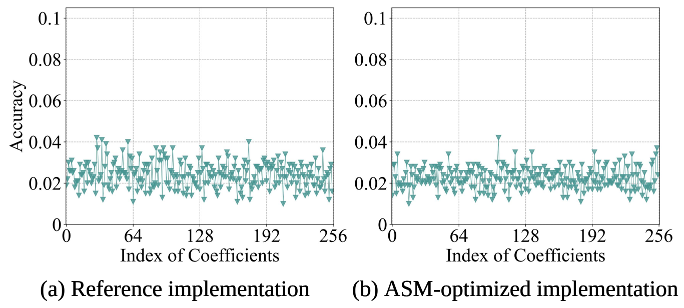


Fig. 11. Side-channel attack accuracy across different coefficients of y.

process and the INTT$(c\mathbf{s}_1)$ process contain four major loops, since both $\mathbf{y}$ and $c\mathbf{s}_1$ are elements of $\mathbb{R}_q^k$, and the operations on them are performed separately on each polynomial component.

Figure 9(a) provides a further zoomed-in view of the trace corresponding to one polynomial component y. With further zooming, we can observe the trace corresponding to a single iteration of the y generation. Figure 9(b) presents a zoomed-in view of the NTT(c) process, where the 8 layers of the NTT can be identified. In the figure, the distinct downward peaks correspond to butterfly units with different twiddle factors. The peaks become denser in the later layers, as these layers involve a greater number of twiddle factors, as illustrated in Figure 7. We apply SCA on the first layer of NTT(c). Figure 9(c) shows a zoomed-in view of the INTT(cs_1) process, which is

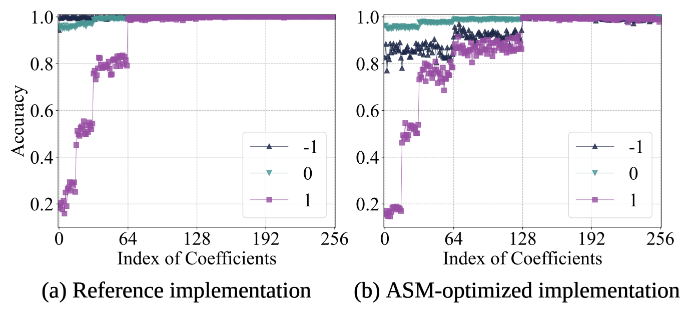


Fig. 12. Side-channel attack accuracy across different coefficients of NTT(c).

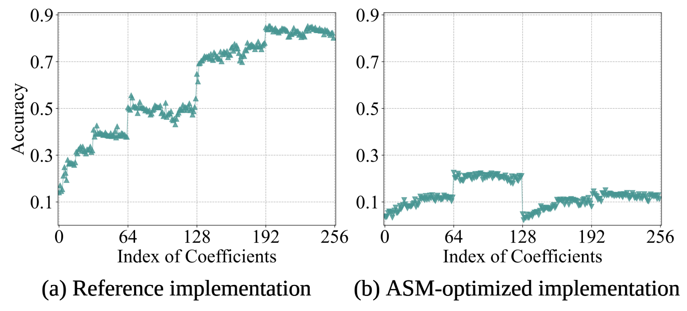


Fig. 13. Side-channel attack accuracy across different coefficients of INTT(cs_1).

roughly the reverse of the process shown in Figure 9(b). The main difference is that, following the first layer, there is an additional multiplication by the scaling factor, corresponding to the multiplication by n^{-1} in Equation 2. This multiplication step is also the target of SCA.

We collected traces from 10000 signing attempts, using 9000 traces as the training set and 1000 traces as the test set. We use template attack combined with linear discriminant analysis as the SCA method. The results for y, c, and cs_1 are shown in Figures 11, 12, and 13, respectively. In these figures, the horizontal axis represents the polynomial coefficient index, and the vertical axis represents the accuracy. We perform the template attack separately on the different coefficients of the polynomials, as the attack accuracy varies across coefficients, as shown in Figures 11, 12, and 13. Using the training set, we identify the coefficients with higher accuracy, and in the test phase, we prioritize the use of these high-accuracy coefficients to improve the overall effectiveness of the attack.

It is worth noting that in Figures 12 and 13, the classification accuracy exhibits a clear dependency on the coefficient index. This phenomenon mainly arises from the inherent structural characteristics of the NTT and INTT. As analyzed in Section IV-C, during each layer of the NTT, the coefficients that are multiplied by rotation factors have significantly higher SNR than those that are not. Consequently, the coefficients involved in such multiplications tend to achieve higher classification accuracy. Figure 12(b) clearly illustrates this relationship: the coefficients indexed from 129 to 256 exhibit higher classification accuracy than those from 1 to 128, which results from whether the coefficients are multiplied by rotation factors in the first layer of the NTT. Similarly, in Figure 12(b), the

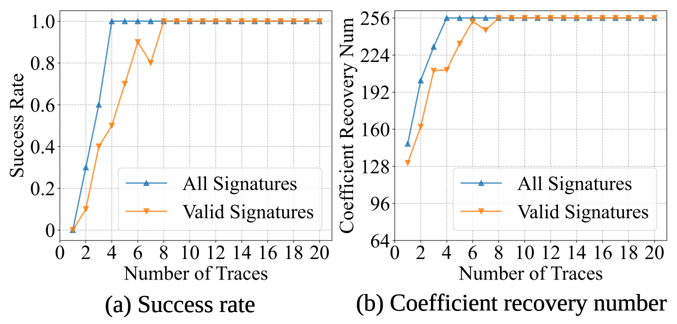


Fig. 14. Private key recovery success rate and number of recovered key coefficients using BP under different numbers of attack traces with reference implementation.

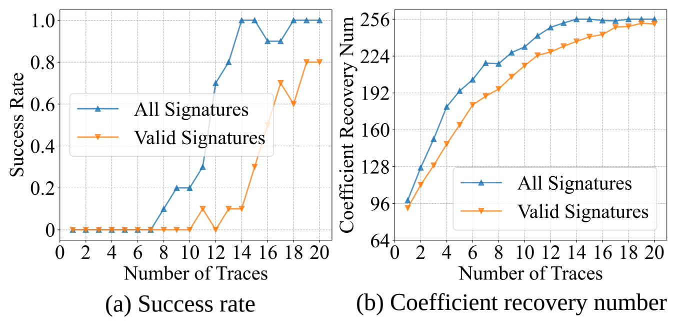


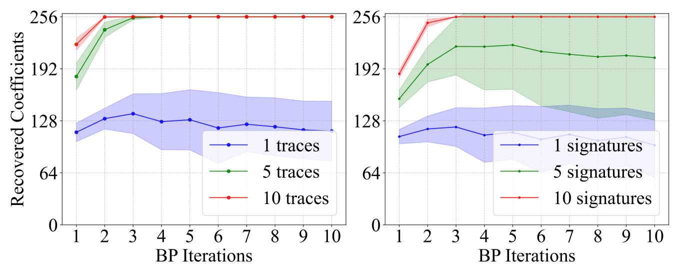
Fig. 15. Private key recovery success rate and number of recovered key coefficients using BP under different numbers of attack traces with ASM-optimized implementation.

coefficients from 65 to 128 show higher accuracy than those from 1 to 64, which corresponds to the multiplication pattern in the second layer of the NTT. This pattern continues through the subsequent layers, leading to the stepwise accuracy distribution observed in Figure 12. Since the INTT has a nearly mirrored structure to the NTT, a similar correlation between coefficient index and accuracy can be observed in Figure 13(a). The distinct behavior shown in Figure 13(b) differs from the other cases, which we attribute to the characteristics of the ASM-optimized implementation. As this aspect is beyond the main focus of this work, it is not discussed further here.

D. BP Results

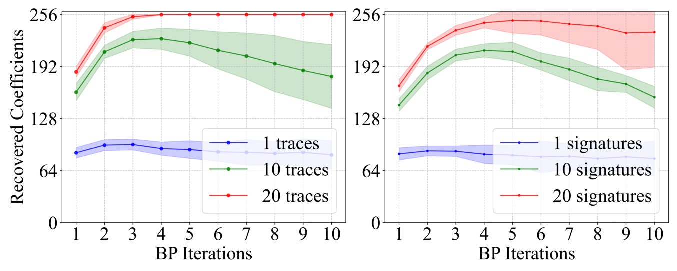
As shown in Figures 14 and 15, the results of the BP algorithm under different number of traces and optimization levels are presented. The horizontal axis represents the number of traces required for the attack, while the vertical axis represents either the success rate or the number of recovered key coefficients. For each trace count, we repeated the experiment 10 times and reported the average results.

The results indicate that with reference implementation, using all signatures (both valid and rejected), we require a minimum of 2 and a maximum of 4 traces for full key recovery. In contrast, using only valid signatures, it takes a minimum of 2 and a maximum of 8 traces for full key recovery. The use of all signatures decreases the number of traces by approximately 50.0% for achieving a 100% key recovery success rate, compared to using only valid signatures.



(a) All signatures (b) Valid signatures

Fig. 16. Recovered coefficients number in BP iteration with reference implementation.



(a) All signatures (b) Valid signatures

Fig. 17. Recovered coefficients number in BP iteration with ASM-optimized implementation.

With ASM-optimized implementation, using all signatures, the attack requires a minimum of 8 and a maximum of 14 traces for full key recovery. When using only valid signatures, the attack requires a minimum of 11 and a maximum of 20+ traces. In this case, leveraging all signatures decreases the number of traces by approximately 30.0% over using only valid signatures.

As shown in Figures 16 and 17, the variation in the number of recovered key coefficients during the BP iteration is depicted for different trace counts and implementation type, where the color represents the average number of recovered key coefficients.

It can be observed that the BP algorithm converges from the 2-th to 5-th iterations. If the key coefficients can be fully recovered, this typically occurs within a maximum of 5 iterations. However, BP does not always converge in a favorable direction. When fewer signing attempts are used in the attack, the number of recovered key coefficients may initially increase but then decrease and converge to a very low value, as illustrated in the case where only a single trace is used.

As shown in Table III, the time and memory overhead of the BP algorithm vary with the number of signatures used.

TABLE III
TIME AND MEMORY OVERHEAD OF THE BP ALGORITHM

Number of Signatures	10	100	1000	5000	10000
Time Per Iteration (s)	1.5	14.6	148	743	1495
Memory (MB)	231	339	1440	6323	12426

On average, each signature requires approximately 1.5 seconds per BP iteration. Additionally, each additional signature incurs an extra memory overhead of approximately 1.2 MB. These data were collected under a single-core setup. With multi-core optimization, the time required per iteration can be further reduced. Considering that each signing attempt typically produces around four signatures, to fully recover the coefficients of s_1, the average BP iteration time required is approximately 288 seconds. The corresponding memory overhead is 240 MB.

Discussion. To better interpret the above BP results, we briefly analyze the expected number and effective influence of rejected signatures in our setting. According to the ML-DSA standard and our measurements in Table II, the expected number of rejected signatures per signing attempt at the targeted parameter sets is about 3.25. In an idealized scenario where the challenge c is recovered with 100% accuracy, a rejected signature would provide essentially the same amount of information as a valid one under our leakage model. One would then expect that jointly exploiting valid and rejected signatures could reduce the required number of traces by roughly 75%, since each signing attempt would on average yield $1 + 3.25$ useful signatures instead of only one.

In practice, however, the accuracy of the recovered c is strictly below 100%, and our experiments in Section V-E show that the attack performance is extremely sensitive to this accuracy. Incorporating rejected signatures associated with low-accuracy estimates of c tends to mislead the BP inference: even a small number of such “bad” rejected signatures can drive the messages towards an incorrect mode, from which the subsequent addition of correctly modeled hints is often insufficient to recover. For this reason, in our BP experiments we do not blindly use all rejected signatures. Instead, we apply a simple filtering rule and discard rejected traces for which the reconstructed c has a number of ± 1 coefficients that deviates significantly from the target weight τ. As a result, only a subset of the available rejected signatures can be safely exploited, and the empirical reduction in trace complexity is smaller than the theoretical 75%: in our experiments, using all (filtered) signatures reduces the required number of traces by about 50% for the reference implementation and about 30% for the ASM-optimized implementation. These observations highlight that the accuracy of c is a key bottleneck for effectively leveraging rejected signatures, which motivates the dedicated analysis of c-recovery accuracy in the next subsection.

E. Impact of c Recovery Accuracy on Key Recovery

For a clearer illustration of how the accuracy of c affects the attack performance, we have added an additional set of simulation experiments. In the experiments, c's accuracy was artificially controlled, and only its hard information was used in the BP phase, while all other data came from real measurements. The results, shown in Figures 18 and 19, correspond to the reference and ASM-optimized implementations, respectively. As can be seen, when the average accuracy of c drops to 0.95, the key can still be recovered in the reference implementation, albeit with a significantly higher number of traces required—thanks to its higher SNR on cs_1.

In contrast, for the ASM-optimized implementation, recovery fails completely once the accuracy falls below 0.97, due to its lower SNR. Overall, these findings demonstrate that the accuracy of c has a substantial impact on the success of key recovery, particularly when the SNR of cs_1 is low. Achieving a sufficiently high recovery accuracy for c is therefore crucial for a successful attack.

As a reference, in our real experiments, the average accuracy of c was 97.5% for the reference implementation and 95.3% for the ASM-optimized implementation. By comparing the "All Signatures" results in Figures 16 and 17 with those in Figures 18 and 19, it can be observed that by using the soft information of c for key recovery, our method achieved performance equivalent to that obtained with pure hard information at 99% accuracy in the reference implementation, and between 98% and 99% accuracy in the ASM-optimized implementation. By comparing the "Valid Signatures" results in Figures 16 and 17 with those in Figures 18 and 19, it can be observed that when the accuracy of c is not sufficiently high ($\leq$ 97% for the reference implementation and $\leq$ 98% for the ASM-optimized implementation), incorporating additional information from rejected signatures actually degrades the overall attack performance, resulting in worse outcomes than using only valid signatures.

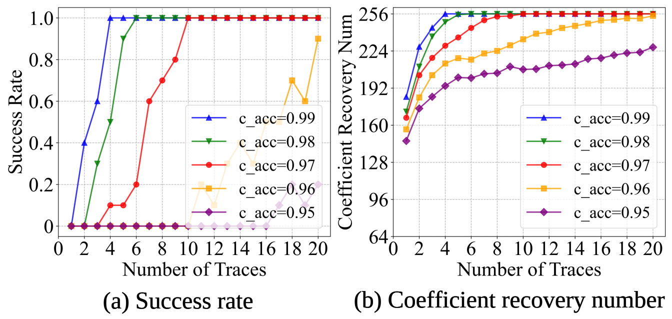


Fig. 18. Impact of c accuracy on key recovery success rate and average number of recovered key coefficients when both valid and rejected signatures are used (reference implementation).

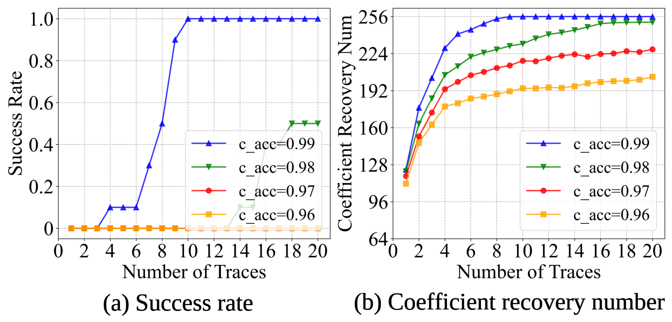


Fig. 19. Impact of c accuracy on key recovery success rate and average number of recovered key coefficients when both valid and rejected signatures are used (ASM-optimized implementation).

To better understand the relative information content of different leakages related to rejected signatures, we also provide experimental results for the case where only rejected signatures are used. Specifically, we add a simulation experiment that evaluates the key-recovery efficiency under varying accuracies of c when only rejected signatures are available. In this experiment, the accuracy of c is generated through simulation, and only the hard information of c is employed, while all other data and attack parameters are identical to those in the real experiments. As shown in Figures 20 and 21, the attack success rate drops rapidly for both implementations as the accuracy of c decreases. Our method requires fewer than 30 traces to recover the key when the accuracy of c reaches 98% in the reference implementation or 99% in the ASM-optimized implementation. For comparison, in scenarios where only the rejection behavior or coarse information related to the response z is exploited, the evaluations in [17], [18] report that about 10^6 rejected signatures (approximately 10^5 traces) are needed for full key recovery even when c is known with 100% accuracy. This suggests that, when one restricts the analysis to z-based rejection information, the per-signature information content is much lower than in our setting with leakages on $\mathrm{INTT}(cs_1)$.

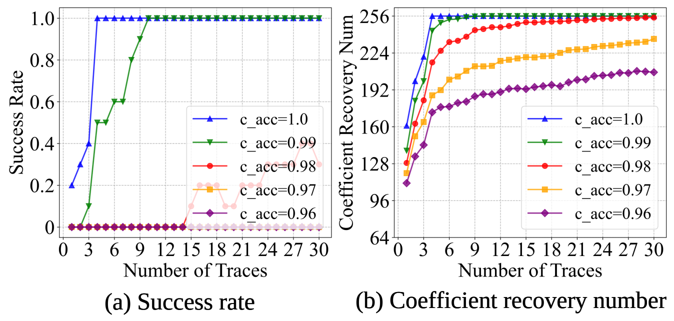


Fig. 20. Impact of c accuracy on key recovery success rate and average number of recovered key coefficients when only rejected signatures are used (reference implementation).

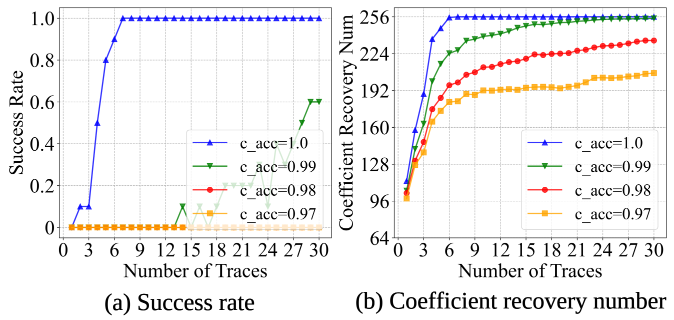


Fig. 21. Impact of c accuracy on key recovery success rate and average number of recovered key coefficients when only rejected signatures are used (ASM-optimized implementation).

F. Comparison With Existing Works

As shown in Table IV, a comparison is provided between this work and existing studies. Although some previous works [14], [19]–[21] have reported single-trace key-recovery results, our work remains meaningful, as those studies are subject to various limitations. First, the attack scenarios in [19]–[21] are

TABLE IV
COMPARISON WITH EXISTING WORKS

Methods	Applicable Scenarios	Experimental Type	Implementation Type	Optimization Level	Main Method[†]	Utilizing Rejected Signatures	Error Tolerance	Number of Traces Required[*]
[18]	General	Simulation	-	-	TA, BP	No	High	4
[18]	General	Simulation	-	-	TA, BP	Yes (z-based)	High	10^5
[17]	General	Practical	Reference	-	DL, ILP	Inefficient	Medium	10^6
[14]	General	Practical	Reference	-	DL, ILP	No	Medium	1 (2)
[15]	General	Practical	Reference	O3	DL, ILP	No	Medium	10^5
[16]	General	Practical	Reference	O3	TA, LSM	No	Medium	10^6
[19]	Constrained	Practical	ASM-optimized	O3	DL	No	Low	1 (100)
[20]	Constrained	Practical	Reference	O3	DL	No	Low	1 (1)
[21]	Constrained	Practical	Reference	O3	DL	No	Low	1 (1)
This Work	General	Practical	Reference & ASM-optimized	O3	TA, BP	**Efficient**	**High**	**2 (4)** & **8 (14)**

[†] TA stands for the template attack.
[*] The value outside the parentheses represents the minimum required traces, while the value inside the parentheses indicates the number of traces required for a 100% success rate. For larger results, we only present their approximate order of magnitude.

restricted: they rely on direct attacks against operations such as key unpacking or NTT computations. Such operations may not be available or repeated in typical deployment settings (for example, keys are unpacked or transformed only at device startup), which reduces the practical applicability of these attacks. Second, the successes reported in [14], [20], [21] depend on high SNR values obtained from the reference implementation; these studies do not examine the more realistic, lower-SNR behavior that appears in ASM-optimized implementations. Although [19] reports single-trace recovery even for an ASM-optimized implementation, the number of traces required to guarantee 100% recovery in [19] is much larger than in our work (100 versus 14). Third, [14], [19]–[21] all rely on deep-learning based classifiers, which substantially increases the modeling cost in the modeling phase.

By contrast, our work targets a more general and practically relevant scenario: we focus on operations that are executed for every signing attempt, reduce reliance on very high SNR, and relax the attacker's modeling requirements. In particular, when single-trace recovery becomes infeasible under such general conditions, our method exploits information from both valid and rejected signatures to substantially reduce the number of traces required for successful key recovery.

Regarding the use of rejected signatures, Zhou et al. [17] focus on leakage related to the rejected response z and its corresponding challenge c. Their attack exploits the values of rejected z together with c within an ILP-based framework, and relies solely on rejected signatures. Under this leakage model, about 10^6 rejected signatures (roughly 10^5 traces) are required to recover the secret key, which suggests that the information extracted from each rejected signature is still relatively limited in practice.

Bronchain et al. [18] propose a generic SASCA framework for ML-DSA and explicitly remark that both accepted and rejected coefficients could, in principle, be exploited within the same factor graph. However, their work does not provide an implementation-level evaluation of such combined use on a concrete embedded implementation, nor does it analyze the practical efficiency and feasibility of exploiting richer leakages such as those on $\mathrm{INTT}(cs_1)$ and $\mathrm{NTT}(c)$, or the impact of the accuracy of c when rejected signatures are involved.

In this work, we instantiate a BP-based attack that exploits leakages on $\mathrm{INTT}(cs_1)$ and $\mathrm{NTT}(c)$ to make practical use of both valid and rejected signatures on a Cortex-M4 implementation of ML-DSA. We provide implementation-level evaluations for different scenarios, including using all signatures and using only rejected signatures, and show that fewer than 30 traces are sufficient to recover the key in the rejected-only case. We further study how the accuracy of c affects the attack performance and evaluate a BP message-update rule that incorporates soft information of c, demonstrating that it can effectively mitigate the performance degradation caused by imperfect c recovery. These results complement existing works by giving a concrete feasibility and efficiency analysis of exploiting rejected signatures under realistic leakage and noise conditions.

VI. CONCLUSION AND FUTURE WORK

This paper presents a practical and efficient BP-based attack on ML-DSA that makes use of the side-channel leakages generated during a single signing attempt. Building on the generic SASCA framework for ML-DSA, we instantiate an ML-DSA-specific factor-graph construction that incorporates leakages on $\mathrm{INTT}(cs_1)$, $\mathrm{NTT}(c)$, and both valid and rejected signatures within a unified BP instance. We further provide practical guidance on applying BP to ML-DSA, addressing key implementation considerations such as leakage modeling, message-update strategies, and the treatment of imperfectly recovered challenges. Our implementation-level evaluation on a Cortex-M4 platform shows that rejected signatures can be exploited far more effectively when richer leakages such as $\mathrm{INTT}(cs_1)$ are available, and that combining valid and rejected signatures can substantially reduce the number of traces required for full key recovery. We additionally analyze the impact of the accuracy of c on the attack performance and demonstrate that incorporating soft information of c can significantly mitigate the degradation caused by imperfect recovery of c.

For protected implementations, to the best of our knowledge, the state-of-the-art masking countermeasures for ML-

DSA do not cover the generation/handling of the challenge c, meaning that the challenge c remains recoverable. This ensures that the core idea of this work—leveraging rejected signatures for enhanced key recovery—remains practical and applicable even in the presence of countermeasures. However, under countermeasures, performing SCA on y and cs_1 becomes significantly more challenging, which may reduce the overall feasibility of the proposed method. In future work, we plan to further investigate how to effectively exploit rejected signatures for private key recovery in protected implementations of ML-DSA.

Acknowledgment

The authors would like to thank the editor and the anonymous reviewers for their insightful comments and constructive suggestions, which have helped to significantly improve the quality and clarity of this paper.

While preparing the final version of this work, we became aware of two concurrent and independent studies. Zhou et al. [17] investigate key recovery for CRYSTALS-Dilithium from leakage in the rejection sampling procedure, and in particular propose methods to recover the challenge c from side-channel leakage on $\mathrm{NTT}(c)$. Our treatment of recovering c from leakage on $\mathrm{NTT}(c)$ follows a similar line of reasoning in the ML-DSA setting. In addition, Sun et al. [29] independently analyze how many bits of the vector y are sufficient to construct effective hints, arriving at conclusions that are closely related to our bit-length analysis for y in ML-DSA. We gratefully acknowledge these concurrent contributions, which further underline the importance of studying side-channel leakage in lattice-based signature schemes.

References

[1] P. C. 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*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_25

[2] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2002, pp. 13–28. [Online]. Available: https://doi.org/10.1007/3-540-36400-5_3

[3] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, "Side-channel assisted existential forgery attack on dilithium - A NIST PQC candidate," *IACR Cryptol. ePrint Arch.*, p. 821, 2018. [Online]. Available: https://eprint.iacr.org/2018/821

[4] Z. Qiao, Y. Liu, Y. Zhou, M. Shao, and S. Sun, "When NTT meets SIS: efficient side-channel attacks on dilithium and kyber," *IACR Cryptol. ePrint Arch.*, p. 1866, 2023. [Online]. Available: https://eprint.iacr.org/2023/1866

[5] H. M. Steffen, G. Land, L. J. Kogelheide, and T. Güneysu, "Breaking and protecting the crystal: Side-channel analysis of dilithium in hardware," in *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023, College Park, MD, USA, August 16-18, 2023, Proceedings*, ser. Lecture Notes in Computer Science, T. Johansson and D. Smith-Tone, Eds., vol. 14154. Springer, 2023, pp. 688–711. [Online]. Available: https://doi.org/10.1007/978-3-031-40003-2_25

[6] H. Wang, Y. Gao, Y. Liu, Q. Zhang, and Y. Zhou, "In-depth correlation power analysis attacks on a hardware implementation of crystals-dilithium," *Cybersecur.*, vol. 7, no. 1, p. 21, 2024. [Online]. Available: https://doi.org/10.1186/s42400-024-00209-9

[7] Y. Liu, Y. Liu, Y. Zhou, Y. Gao, Z. Qiao, and H. Wang, "A novel power analysis attack against crystals-dilithium implementation," in *IEEE European Test Symposium, ETS 2024, The Hague, Netherlands, May 20-24, 2024*. IEEE, 2024, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ETS61313.2024.10567325

[8] T. Tosun and E. Savas, "Zero-value filtering for accelerating non-profiled side-channel attack on incomplete ntt-based implementations of lattice-based cryptography," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 3353–3365, 2024. [Online]. Available: https://doi.org/10.1109/TIFS.2024.3359890

[9] T. Tosun, A. Moradi, and E. Savas, "Exploiting the central reduction in lattice-based cryptography," *IEEE Access*, vol. 12, pp. 166 814–166 833, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3494593

[10] A. P. Fournaris, C. Dimopoulos, and O. G. Koufopavlou, "Profiling dilithium digital signature traces for correlation differential side channel attacks," in *Embedded Computer Systems: Architectures, Modeling, and Simulation - 20th International Conference, SAMOS 2020, Samos, Greece, July 5-9, 2020, Proceedings*, ser. Lecture Notes in Computer Science, A. Orailoglu, M. Jung, and M. Reichenbach, Eds., vol. 12471. Springer, 2020, pp. 281–294. [Online]. Available: https://doi.org/10.1007/978-3-030-60939-9_19

[11] Z. Chen, E. Karabulut, A. Aysu, Y. Ma, and J. Jing, "An efficient non-profiled side-channel attack on the crystals-dilithium post-quantum signature," in *39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24-27, 2021*. IEEE, 2021, pp. 583–590. [Online]. Available: https://doi.org/10.1109/ICCD53106.2021.00094

[12] Z. Qiao, Y. Liu, Y. Zhou, J. Ming, C. Jin, and H. Li, "Practical public template attack attacks on crystals-dilithium with randomness leakages," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1–14, 2023. [Online]. Available: https://doi.org/10.1109/TIFS.2022.3215913

[13] Y. Liu, Y. Zhou, S. Sun, T. Wang, R. Zhang, and J. Ming, "On the security of lattice-based fiat-shamir signatures in the presence of randomness leakage," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1868–1879, 2021. [Online]. Available: https://doi.org/10.1109/TIFS.2020.3045904

[14] Z. Qiao, Y. Liu, Y. Zhou, Y. Zhao, and S. Chen, "Single trace is all it takes: Efficient side-channel attack on dilithium," *IACR Cryptol. ePrint Arch.*, p. 512, 2024. [Online]. Available: https://eprint.iacr.org/2024/512

[15] S. Marzougui, V. Ulitzsch, M. Tibouchi, and J. Seifert, "Profiling side-channel attacks on dilithium: A small bit-fiddling leak breaks it all," *IACR Cryptol. ePrint Arch.*, p. 106, 2022. [Online]. Available: https://eprint.iacr.org/2022/106

[16] A. Berzati, A. C. Viera, M. Chartouny, S. Madec, D. Vergnaud, and D. Vigilant, "Exploiting intermediate value leakage in dilithium: A template-based approach," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 4, pp. 188–210, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i4.188-210

[17] Y. Zhou, W. Wang, Y. Sun, and Y. Yu, "Rejected signatures' challenges pose new challenges: Key recovery of crystals-dilithium via side-channel attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 4, pp. 817–847, 2025. [Online]. Available: https://doi.org/10.46586/tches.v2025.i4.817-847

[18] O. Bronchain, M. Azouaoui, M. ElGhamrawy, J. Renes, and T. Schneider, "Exploiting small-norm polynomial multiplication with physical attacks application to crystals-dilithium," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 2, pp. 359–383, 2024. [Online]. Available: https://doi.org/10.46586/tches.v2024.i2.359-383

[19] R. Wang, K. Ngo, J. Gärtner, and E. Dubrova, "Single-trace side-channel attacks on crystals-dilithium: Myth or reality?" *IACR Cryptol. ePrint Arch.*, p. 1931, 2023. [Online]. Available: https://eprint.iacr.org/2023/1931

[20] J. Han, T. Lee, J. Kwon, J. Lee, I. Kim, J. Cho, D. Han, and B. Sim, "Single-trace attack on NIST round 3 candidate dilithium using machine learning-based profiling," *IEEE Access*, vol. 9, pp. 166 283–166 292, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3135600

[21] I. Kim, T. Lee, J. Han, B. Sim, and D. Han, "Novel single-trace ML profiling attacks on NIST 3 round candidate dilithium," *IACR Cryptol. ePrint Arch.*, p. 1383, 2020. [Online]. Available: https://eprint.iacr.org/2020/1383

[22] N. I. o. S. U.S. Department of Commerce and Technology, "Fips 204: Module-lattice-based digital signature standard (ml-dsa)," National Institute of Standards and Technology, Tech. Rep. FIPS 204, August 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.204

[23] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai, "Crystals-dilithium," *Algorithm Specifications and Supporting Documentation*, 2020.

[24] A. Satriawan and R. Mareta, "A complete beginner guide to the number theoretic transform (NTT)," *IACR Cryptol. ePrint Arch.*, p. 585, 2024. [Online]. Available: https://eprint.iacr.org/2024/585

[25] N. Veyrat-Charvillon, B. Gérard, and F. Standaert, "Soft analytical side-channel attacks," in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8873. Springer, 2014, pp. 282–296. [Online]. Available: https://doi.org/10.1007/978-3-662-45611-8_15

[26] C. Knoll, "Understanding the behavior of belief propagation," *CoRR*, vol. abs/2209.05464, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2209.05464

[27] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 513–533. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4_25

[28] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, ser. Lecture Notes in Computer Science, P. Schwabe and N. Thériault, Eds., vol. 11774. Springer, 2019, pp. 130–149. [Online]. Available: https://doi.org/10.1007/978-3-030-30530-7_7

[29] Y. Sun, T. Zhang, Z. Huang, Y. Yu, Y. Zhuang, S. Sun, and W. Wang, "Finding more hints - improved power analysis attacks on dilithium," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 11 963–11 974, 2025. [Online]. Available: https://doi.org/10.1109/TIFS.2025.3618387