

# Efficient Cpa Attack on Hardware Implementation of Ml-Dsa in Post-Quantum Root of Trust

Publisher: IEEE

Cite This

PDF

Merve Karabulut ; Reza Azarderakhsh All Authors

1

Cites in Paper

287

Full Text Views



Abstract

Document Sections

I. Introduction

II. Background

III. The Proposed Side-Channel Attack

IV. Implementation-Level Analysis

V. Evaluation Result

Show Full Outline

Authors

Figures

References

Citations

Keywords

Metrics

More Like This

Footnotes

**Abstract:**

Side-channel attacks (SCA) pose a significant threat to cryptographic implementations, including those designed to withstand the computational power of quantum computers. This paper introduces the first side-channel attack on an industrygrade post-quantum cryptography implementation. Specifically, we present a Correlation Power Analysis (CPA) attack targeting the open-source hardware implementation of ML-DSA within a Silicon Root of Trust framework developed through a multi-party collaboration involving leading technology companies. Our attack focuses on the modular reduction process that follows the Number Theoretic Transform-based polynomial pointwise multiplication. By exploiting side-channel leakage from a distinctive unique reduction algorithm and leveraging the zeroization mechanism used to securely erase sensitive information by clearing internal registers, we significantly enhance the attack's efficacy. Our findings reveal that an adversary can extract the secret keys using only 10,000 power traces. With access to these keys, an attacker could forge signatures for certificate generation, thereby compromising the integrity of the root of trust. This work highlights the vulnerabilities of industry-standard root-oftrust systems to side-channel attacks. It underscores the urgent need for robust countermeasures to secure commercially deployed systems against such threats.

Published in: 2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)

Date of Conference: 05-08 May 2025

DOI: 10.1109/HOST64725.2025.11050056

Date Added to IEEE Xplore: 07 July 2025

Publisher: IEEE

ISBN Information:

Conference Location: San Jose, CA, USA

ISSN Information:

Funding Agency:

## SECTION I. Introduction

Quantum computing presents a significant challenge to current classical cryptosystems based on the computational difficulty of problems like integer factorization [1] and discrete logarithms [2], [3]. Shor's algorithm demonstrates that these problems are vulnerable to quantum computers, which can tackle their complexity in polynomial time [4]. To address this threat, the National Institute of Standards and Technology (NIST) initiated the post-quantum cryptography (PQC) standardization process [5], evaluating multiple candidates for digital signatures and key encapsulation algorithms. CRYSTALS-Dilithium, Falcon, and SPHINCS+ were selected as standards for digital signatures, with CRYSTALS-Dilithium standing out for its robustness and adaptability. NIST has officially renamed CRYSTALS-Dilithium as the Modular Lattice Digital Signature Algorithm (ML-DSA), a term used consistently throughout this paper [6].

ML-DSA is a lattice-based digital signature scheme derived from the Fiat-Shamir paradigm [6], with security rooted in the computational difficulty of finding short vectors in lattices. It combines strong security guarantees with operational efficiency [7]. However, like many cryptographic schemes, its implementations might be vulnerable to side-channel attacks [8]–[26], which exploit information leakage through power consumption, electromagnetic emissions, and other observable channels. Addressing these vulnerabilities is critical, as side-channel resilience is essential for secure use in realworld applications.

Given the critical nature of cryptographic operations, prior research focuses extensively on analyzing and exploiting sidechannel vulnerabilities [8]–[22] to understand their implications and propose mitigations [27]–[31]. Most of these studies target software implementations [8]–[20], as they are more accessible for evaluation and frequently employed in earlystage algorithm testing. While post-quantum cryptography can be implemented in software, the computationally intensive structure of the ML-DSA algorithm makes hardware implementations the preferred choice for security-critical applications such as Root of Trust (RoT) [32]. Prior research on software implementations often focuses on reference designs that are unrepresentative of deployed systems.

This leaves a critical gap in pinpointing the vulnerabilities of hardware implementations, particularly within widely adopted cryptographic frameworks. In contrast to extensive research on software implementations, studies targeting hardware sidechannel attacks remain comparatively limited [21], [22]. Existing works, such as those by Steffen et al. [21] and Wang et al. [22], focus primarily on standalone operations in ML-DSA or CRYSTALS-Dilithium hardware implementation, which are not aligned with NIST's Federal Information Processing Standards (FIPS) 204 standardization [6]. Notably, none of these works target post-quantum cryptographic implementations used in industry-level applications.

This work addresses a critical gap by targeting a cryptographic Intellectual Property (IP) [34], an implementation of ML-DSA, within a Silicon Root of Trust framework [32]. This collaborative framework, developed by leading technology companies, is designed to enhance trust in modern System-on-Chip (SoC) platforms. **It is important to note that the implementation analyzed in this work represents an early, work-in-progress version, which continues to undergo updates and has not yet been finalized.** We present the first side-channel attack on a post-quantum cryptography implementation used in an industry-standard setting. Our approach achieves a 99.99 % success rate using only 10,000 power traces, significantly outperforming prior works. [Table 1](#) highlights these advancements, comparing our results with previous hardware attacks on CRYSTALS-Dilithium and MLDSA. For example, Steffen et al. achieved a 94.2 % success rate in profiling attack on Number Theoretic Transform (NTT) operation using 350,000 traces and a 93.2 % success rate in decoding with 50,000 traces. Similarly, Wang et al. conducted a CPA attack that required 70,000 traces.

**Table I:** Comparison of hardware side-channel attacks on crystals-dilithium and ML-DSA

Work	Type	Implementation	Target Op.	Traces ( $\times 10^3$ )	Success Rate	Real-World App.
[21]	P	Dilithium [33]	NTT	350	94.2%	✗
[21]	P	Dilithium [33]	Decoding	50	93.2%	✗
[21]	NP	Dilithium [33]	Pointwise Mult.	66	–	✗
[22]	NP	Dilithium [33]	Pointwise Mult.	70	–	✗
<b>This work</b>	<b>NP</b>	<b>ML-DSA [34]</b>	<b>Pointwise Mult.</b>	<b>10</b>	<b>99.99%</b>	<b>✓</b>

P = Profiling, NP = Non-Profiling, Target Op. = Target Operation, Real-World App. = Real-World Application

Our non-profiling approach specifically targets the ML-DSA signing process, eliminating the need for identical devices or pre-generated templates. This attack effectively exposes vulnerabilities in the implementation of unprotected ML-DSA hardware, emphasizing the critical need for robust countermeasures in post-quantum cryptography implementations. The contributions of this paper are summarized as follows:

- We present the first side-channel attack targeting the hardware implementation of ML-DSA [34], an open-source design intended for adoption in hardware platforms by major technology companies such as the leading technology companies. Our findings reveal a critical vulnerability in a widely used hardware design.
- We target the NTT-based pointwise multiplication operation, identifying and exploiting previously unknown vulnerabilities. These vulnerabilities allow us to recover partial secret key coefficients, which are integral to the certificate generation process within the Silicon RoT.

We analyze and identify a vulnerability in the reduction algorithm implemented in the studied hardware [34], designed specifically for the modulus  $q=8380417 = 2^{23} \cdot 2^{13} + 1$ . Unlike commonly used reduction techniques such as Barrett [35] or Montgomery [36] algorithms, this implementation leverages a particular structure of the modulus to optimize performance, inadvertently introducing exploitable side-channel leakage.

- We demonstrate the efficiency of our side-channel attack by successfully recovering partial secret key coefficients with a 99.99 % success rate using only 10,000 power traces. This significantly improves on existing attacks, which need far more traces for comparable or lower success rates.

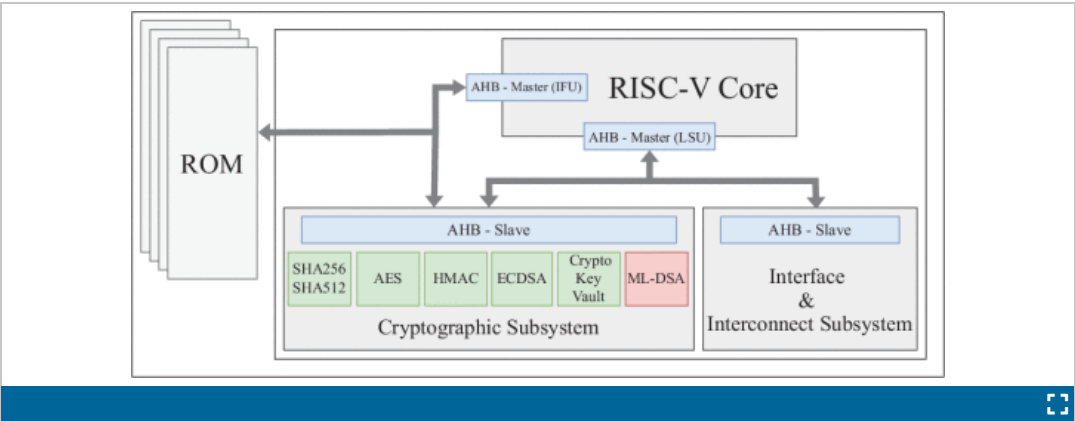


Fig. 1:

High level of the RoT, which employs classical and PQC digital signatures for authentication [32].

---

## SECTION II. Background

This section provides an overview of open-source implementation of ML-DSA [34] within the silicon RoT internal block [32], focusing on its parameter settings and operations involving the secret key, which are of interest to adversaries. Additionally, it introduces the adversarial capabilities and outlines the general CPA methods used to recover the secret key.

### A. Notations

Let  $n$  and  $q$  be two integers, where  $n = 256$  and  $q = 8380417$ . We define  $R_q$  as the polynomial ring  $\mathbb{Z}[x]/(x^n + 1)$ . The infinity norm  $|x|_\infty$  denotes the maximum absolute value among all coefficients of a polynomial  $x$ .

Matrices are represented by bold capital letters, such as  $\mathbf{A} \in R_q^{k \times \ell}$ . Vectors are represented by bold lowercase letters, such as  $\mathbf{y}$ . For polynomial vector  $\mathbf{s}_1 \in R_q^\ell$ , it is pseudorandomly sampled from the set of  $\ell$ -dimensional polynomial vectors, where each coordinate polynomial has short coefficients within the range  $[-\eta, \eta]$ . Polynomials in the NTT domain [7] are denoted with a hat (e.g.,  $\hat{c}$ , where  $\hat{c} = \text{NTT}(c)$ ). This notation is transitive, so  $\hat{\mathbf{s}}$  represents each polynomial in  $\mathbf{s}$  being individually transformed into the NTT domain. The symbol “o” denotes pointwise multiplication, and “ $\perp$ ” indicates an invalid or undefined value. The set  $B_\tau$  represents all possible challenge polynomials with exactly  $\tau$  coefficients of  $\pm 1$ , and the rest 0. Lastly,  $H$  represents the SHAKE algorithm, as defined in [6]. This cryptographic hash function is used for key derivation and signature binding operations throughout the ML-DSA signature scheme.

### B. Root of Trust

This hardware security module provides essential RoT capabilities designed for integration into SoC systems used in cloud platforms [32]. Figure 1 illustrates RoT block's highlevel architecture, which consists of a RISC-V core, ROM, cryptographic subsystem, and an interface & interconnect subsystem, all interconnected via the AHB (Advanced HighPerformance Bus) protocol. These components collaboratively enable secure device identity and hierarchical key derivation. This architecture ensures robust protection against firmware compromise while facilitating secure initialization and reliable communication within the SoC system.

**Table II:** Parameter sets of ML-DSA [6]

---

Security Level	ML-DSA-44	ML-DSA-65	ML-DSA-87
Parameter	Values		
$q$	8380417	8380417	8380417
$d$	13	13	13
$\tau$	39	49	60
$\lambda$	128	192	256
$\gamma_1$	$2^{17}$	$2^{19}$	$2^{19}$
$\gamma_2$	$\frac{q-1}{88}$	$\frac{q-1}{32}$	$\frac{q-1}{32}$
$(k, \ell)$	(4,4)	(6,5)	(8,7)
$\eta$	2	4	2
$\beta = \tau \cdot \eta$	78	196	120
$\omega$	80	55	75

The ROM provides immutable storage for the RoT code, serving as the foundation for secure initialization and attestation. It is directly linked to the RISC-V core, which features two AHB master interfaces: the Instruction Fetch Unit (IFU) for fetching instructions and the Load/Store Unit (LSU) for data transfers. These interfaces interact with the cryptographic subsystem and the interface and interconnect subsystem via AHB slave connections. The cryptographic subsystem supports hashing (SHA-256/SHA-512), encryption (AES), keyed-hash authentication (HMAC), elliptic curve signatures (ECDSA), and a classical cryptographic key vault [32].

In addition to classical cryptography, the subsystem integrates ML-DSA, a post-quantum digital signature scheme, to provide quantum resistance. The hardware security module uses ML-DSA-87 in combination with ECC Secp384r1 to implement a dual-signature scheme for verifying firmware integrity, ensuring a security level of approximately 192 bits for ECDSA [37] and level 5 for ML-DSA. By securing services such as secure boot, firmware verification, and key management, ML-DSA is critical in maintaining the integrity of runtime (RT) code and RoT for Measurement (RTM) [32].

However, our results demonstrate that open-source MLDSA implementation introduces vulnerabilities that are exploitable via side-channel attacks. Specifically, we show that an adversary can extract the ML-DSA secret keys used to establish SoC attestation. With these keys, an attacker could forge signatures, compromise firmware integrity, and ultimately undermine the RoT. This attack highlights the critical need for robust countermeasures to secure post-quantum cryptographic implementations within hardware security modules and similar platforms.

### C. Configuring ML-DSA

To better understand the significance and vulnerabilities of open-source ML-DSA implementation in this framework, it is essential to examine ML-DSA's algorithm steps and security properties. ML-DSA is a lattice-based digital signature scheme secure against quantum computers [6]. Its security is based on the hardness of the Module Learning With Errors (ModuleLWE) and Module Short Integer Solution (Module-MSIS) problems [38]. It has been selected by the NIST as the primary algorithm for quantum-secure digital signatures due to its efficiency, robust security properties, and scalability.

#### Algorithm 1 Key Generation [6]

**Input:** Seed  $\xi$

**Output:** Public key  $pk$  and private key  $sk$

1:  $(\rho, \rho', K) \leftarrow H(\xi)$

```

2:   $\mathbf{A} \in R_q^{k \times l} \leftarrow \text{ExpandA}(\rho)$ 

3:   $(\mathbf{s}_1 \in \mathbb{R}_q^\ell, \mathbf{s}_2 \in \mathbb{R}_q^k) \leftarrow \text{ExpandS}(\rho')$   $\triangleright$  Generate secrets.

4:   $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ 

5:   $(\mathbf{t}_1, \mathbf{t}_0) \leftarrow \text{Power2Round}(\mathbf{t}, d)$ 

6:   $pk \leftarrow \text{pkEncode}(\rho, \mathbf{t}_1)$ 

7:   $tr \leftarrow H(pk)$ 

8:   $sk \leftarrow \text{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$   $\triangleright$  Encode the private key.

9:  return  $(pk, sk)$ 

```

### Algorithm 2 Signature Generation [6]

**Input:** Secret key  $sk$ , message  $M$

**Output:** Signature  $\sigma$

```

1:   $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$ 

2:   $\mathbf{A} \in R_q^{k \times \ell} \leftarrow \text{ExpandA}(\rho)$ 

3:   $\mu \leftarrow H(tr|M)$ 

4:   $\rho'' \leftarrow H(K|\mu)$ 

5:   $\kappa \leftarrow 0, (\mathbf{z}, \mathbf{h}) \leftarrow \perp$ 

6:  while  $(\mathbf{z}, \mathbf{h}) = \perp$  do  $\triangleright$  Rejection sampling loop

7:   $\mathbf{y} \leftarrow \text{ExpandMask}(\rho'', \kappa)$ 

8:   $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$ 

9:   $\mathbf{w}_1 \leftarrow \text{HighBits}(\mathbf{w})$ 

10:  $\tilde{c} \leftarrow H(\mu|\mathbf{w}_1)$ 

11:  $c \in B_\tau \leftarrow \text{SampleInBall}(\tilde{c})$ 

12:  $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$   $\triangleright$  Target operation: pointwise multiplication

13:  $r_0 \leftarrow \text{LowBits}(\mathbf{w} - c\mathbf{s}_2)$ 

```

```

14:  if  $|z|_{\infty} \geq \gamma_1 - \beta$  or  $|r_0|_{\infty} \geq \gamma_2 - \beta$  then

15:     $(z, h) \leftarrow \perp$ 

16:  else

17:     $h \leftarrow \text{MakeHint}(-ct_0, w - cs_2 + ct_0)$ 

17:  if  $|ct_0|_{\infty} \geq \gamma_2$  or  $\# \text{ of } 1\text{'s in } h > \omega$  then

18:     $(z, h) \leftarrow \perp$ 

19:  end if

20: end if

21:   $\kappa \leftarrow \kappa + \ell$ 

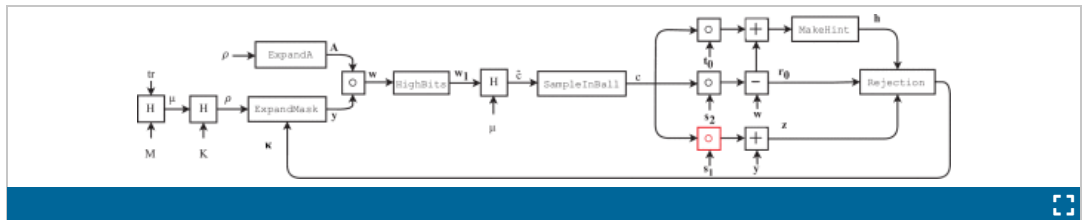
22: end while

23: return  $\sigma = (\tilde{c}, z, h)$ 

```

The parameters in [Table II](#) define the scheme for three security levels, with ML-DSA-87 offering the highest security level. Across all levels, the modulus  $q=8380417$  and the base ring  $R_q = \mathbb{Z}_q[x]/(x^{256} + 1)$  remain constant, while parameters such as  $(k, \ell)$ ,  $\tau$ , and  $\eta$  vary to balance efficiency and security. At higher levels, larger  $\tau$  and  $(k, \ell)$  values are used to enhance robustness, while  $\eta$  determines the coefficient range of secret polynomials. Derived parameters like  $\beta$  and  $\omega$  further influence performance and compliance with constraints. The target design [\[34\]](#) is implemented with the ML-DSA-87 parameters:  $(k, \ell) = (8, 7)$ ,  $\tau = 60$ , and  $\eta = 2$ . These settings provide the strongest cryptographic guarantees within the ML-DSA scheme, ensuring maximum security.

The ML-DSA scheme includes key generation, signature generation, and signature verification. This paper focuses on the key generation and signature generation, as these processes work with secret variables that may be vulnerable to sidechannel attacks. In contrast, the signature verification function does not operate with secret keys and, therefore, is not relevant to this attack. Consequently, this paper does not discuss the signature verification process.



**Fig. 2:**

Illustration of the ML-DSA signing process, highlighting the multiplication of  $c$  and  $s_1$  in red, which is the primary target for side-channel analysis.

The key generation procedure, as defined in [\[6\]](#) and detailed in [Algorithm 1](#), begins with a seed  $\xi$  that generates all cryptographic parameters. Using a hash function  $H$ ,  $\xi$  is expanded into  $\rho, \rho'$ , and  $K \cdot \rho$ .

deterministically generates the public matrix  $\mathbf{A} \in R_q^{k \times l}$  via ExpandA, while  $\rho'$  derives the secret polynomials  $\mathbf{s}_1 \in R_q^\ell$  and  $\mathbf{s}_2 \in R_q^k$ . The public vector  $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ , representing a Module-LWE instance, is split by Power2Round into high-order bits  $\mathbf{t}_1$  (included in the public key) and low-order bits  $\mathbf{t}_0$  (stored in the secret key). The public key is  $pk = (\rho, \mathbf{t}_1)$ , while the private key is  $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$ , providing all components required for signing. This process outputs a compact public key for efficient transmission and storage and a detailed private key for signing operations.

The signature generation procedure, as outlined in [Algorithm 2](#) and illustrated in [Figure 2](#), begins with reconstructing the public matrix  $\mathbf{A} \in R_q^{k \times \ell}$  using the ExpandA function and the seed  $\rho$ . Also, the secret key  $sk$  is decoded to retrieve its components:  $\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2$ , and  $\mathbf{t}_0$ . To bind the signature to the message, the concatenation of  $tr$  and the message  $M$  is hashed using  $H$  to produce  $\mu$ . Another hash operation involving  $K$  and  $\mu$  generates  $\rho''$ , which seeds the pseudorandom masking polynomial  $y$  via the ExpandMask function. The intermediate vector  $\mathbf{w} = \mathbf{A}y$  is then computed, followed by extracting its high-order bits  $\mathbf{w}_1$  using the HighBits function. The challenge polynomial  $c$  is generated by hashing  $\mu$  and  $\mathbf{w}_1$ , and it is subsequently sampled with fixed size  $B_r$  using SampleInBall. The core operation of the signing process is the computation of the response polynomial  $\mathbf{z} = y + c\mathbf{s}_1$ , where  $c\mathbf{s}_1$  represents the pointwise modular multiplication. The polynomial  $c$  and the secret polynomial  $\mathbf{s}_1$  are central to this operation, which is highlighted as the primary target in the proposed attack. This operation is crucial for linking the signature to the secret key, and its details are further discussed in [Section III](#). The modular reduction applied during this step ensures that all coefficients remain within the defined modulus  $q$ . The low-order bits of  $\mathbf{w} - c\mathbf{s}_2$  are extracted as  $r_0$ , which are subjected to rejection sampling to ensure  $\mathbf{z}$  and  $r_0$  meet security constraints. If the constraints are not satisfied, the process resets  $(\mathbf{z}, \mathbf{h})$  and derives a new  $y$ . After satisfying the constraints, the hint polynomial  $\mathbf{h}$  is generated, which compresses low-order bits to enhance verification efficiency. The final signature,  $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ , binds the message  $M$  to the private key components, completing the signing process.

The ML-DSA scheme offers two signing variants: deterministic and hedged. The key distinction lies in the use of randomness during the generation of the commitment value  $y$ . In the hedged variant, fresh randomness is introduced via a Random Bit Generator (RBG) to mitigate side-channel and fault attacks that exploit predictable patterns [\[6\]](#).

Incorporating randomness during the generation of  $y$  would randomize the challenge polynomial  $c$ , disrupting the patterns exploited by side-channel analysis. This would significantly reduce the effectiveness of the proposed attack by obscuring the predictable relationship between power consumption and the processed data. However, the studied implementation [\[34\]](#) is configured to work with ML-DSA deterministic variant. Thus, this implementation creates an exploitable vulnerability.

The deterministic signing approach, adopted in the studied ML-DSA implementation [\[34\]](#), is designed to create stable and predictable cryptographic outputs tied to the device's identity. This approach ensures reproducibility and avoids generating different signatures for the same device under identical conditions, simplifying device identity management and certificate creation [\[39\]](#). However, while this behavior supports specific design objectives, it inadvertently introduces vulnerabilities. Specifically, deterministic signing lacks the fresh randomness required to protect against side-channel attacks [\[6\]](#).

## D. Threat Model

This work adheres to the standard assumptions in nonprofiled power side-channel attacks, where the adversary has physical access to the device and captures power measurements during the execution of the signing operation. The adversary provides input messages and obtains the corresponding signatures, which are known and non-secret information.

In this context, the adversary targets the Root of Trust (RoT) during the certificate generation process. The adversary can provide different firmware payloads as input messages to the signing process, leveraging the system's functionality to generate corresponding certificates. Since the input firmware is known and updatable, the adversary can carefully control the inputs while capturing power traces during the certificate generation operation. Using these power traces, the adversary exploits side-channel leakage to extract critical



secrets, such as private keys, used in the signing process. This aligns with the standard non-profiled power side-channel attack assumptions.

For the proposed differential side-channel attack, we employ the Pearson correlation coefficient as a distinguisher [40]. Unlike profiling attacks, the adversary does not rely on prebuilt templates. Instead, a classical CPA approach is executed to recover the secret key. The CPA process is carried out in the following four steps:

Collect  $n$  power traces during the signing operation, each consisting of  $m$ -length power samples. Store these traces in an  $n \times m$  matrix  $T$ .

Generate an  $n \times k$  intermediate value matrix  $V$  corresponding to all possible key guesses. The intermediate values are calculated based on the controlled input values and the key candidates from the overall key space.

Use the power model to map the intermediate value matrix  $V$  to an  $n \times k$  Hamming Distance matrix  $H$ , where each element  $H_{i,j}$  represents the Hamming Distance corresponding to  $V_{i,j}$ .

Compute the correlation coefficients between the Hamming Distance matrix  $H$  and the actual power consumption matrix  $T$ . Store these coefficients in a correlation matrix  $R_{k \times m}$ . The Pearson correlation coefficient is calculated as:

$$R_{i,j} = \frac{\sum_{x=1}^n (H_{x,i} - \bar{H}_i) (T_{x,j} - \bar{T}_j)}{\sqrt{\sum_{x=1}^n (H_{x,i} - \bar{H}_i)^2 \cdot \sum_{x=1}^n (T_{x,j} - \bar{T}_j)^2}} \quad (1)$$

► View Source ⓘ

Following these steps, the adversary identifies the key guess that maximizes the correlation, thereby recovering the secret key without requiring prior profiling.

### SECTION III.

## The Proposed Side-Channel Attack

ML-DSA is fundamentally based on the Module-LWE problem, which relies on the hardness of recovering secret polynomials from structured linear equations. The cryptographic strength of ML-DSA hinges on maintaining the secrecy of key polynomials such as  $\mathbf{s}_1$ ,  $\mathbf{s}_2$ , and  $\mathbf{t}_0$ . These polynomials are interconnected through relationships derived from the MLDSA construction.

One of the key relationships in ML-DSA can be represented by the equation  $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ , as illustrated in Figure 2, where  $\mathbf{z}$  and  $c$  are known polynomial sets, and  $\mathbf{s}_1$  is a secret polynomial. Before multiplication, the polynomials are transformed into the NTT domain to enable efficient computations, as shown in Figure 3. Extracting  $\mathbf{s}_1$  enables the recovery of  $\mathbf{y}$ , which is used to generate  $\mathbf{w}$ . Once  $\mathbf{w}$  is known, the attacker can compute  $\mathbf{s}_2$  using the outputs of the MakeHint function. Ultimately, with  $\mathbf{s}_1$  and  $\mathbf{s}_2$ , the attacker can solve  $t = A\mathbf{s}_1 + \mathbf{s}_2$ , effectively compromising the Module-LWE foundation of ML-DSA. Consequently, recovering  $\mathbf{s}_1$  provides the attacker with the full secret key of ML-DSA, enabling them to forge arbitrary signatures.

Recovering  $\mathbf{s}_1$  using CPA requires an operation where  $\mathbf{s}_1$  interacts with a known, controllable value, and the interaction must vary based on different inputs to facilitate side-channel analysis. Since the key generation

process produces unique  $\mathbf{s}_1$  values for each input set, it is not suitable for CPA. Instead, we focus on the signature generation process, where  $\mathbf{s}_1$  interacts with a publicly known and attacker-controllable value. As specified in line 12 of [Algorithm 2](#),  $\mathbf{s}_1$  undergoes pointwise multiplication with  $c$ .



**Fig. 3:**

Pointwise multiplication of the polynomial vector  $\mathbf{s}_1$  and the challenge polynomial  $c$  in the NTT domain after transformation. Here,  $\hat{c}$  and  $\hat{\mathbf{s}}_1$  represent the NTT-transformed forms of  $c$  and  $\mathbf{s}_1$ , respectively.

As illustrated in [Figure 2](#),  $c$  is derived during the signature generation process from the SampleInBall function, which uses a hash value  $\tilde{c}$  computed as  $H(\mu|\mathbf{w}_1)$ . Although  $\mathbf{w}$  is a secret value, the high bits of  $\mathbf{w}$ , which is  $\mathbf{w}_1$ , is public value. This sequence ensures that  $c$  is a publicly known and controllable polynomial, making it an ideal candidate for side-channel analysis. This interaction satisfies the criteria for side-channel analysis, making it a highly effective target for recovery.

## SECTION IV.

# Implementation-Level Analysis

This section provides an in-depth examination of the target implementation of [\[34\]](#), with a focus on the NTT implementation block, which is utilized for both NTT transformations and pointwise operations such as modular multiplication, addition, and subtraction. Analyzing this block is critical as it reveals the detailed operation involving the challenge polynomial  $c$  and the secret polynomial  $\mathbf{s}_1$ . Understanding this interaction is essential to pinpointing the vulnerabilities in the implementation and identifying the specific stages where sensitive information may leak through side-channel channels.

In ML-DSA, the polynomials are represented in the NTT domain to optimize computational efficiency during polynomial arithmetic. The use of the NTT domain enables faster computations, particularly for polynomial multiplications. NTT transforms polynomials from their coefficient representation into a point-value representation, enabling polynomial multiplication as simple element-wise multiplication. This transformation relies on modular arithmetic and carefully selected roots of unity. After computations in this transformed domain, the inverse NTT (INTT) is applied to return the result to the coefficient domain. The NTT architecture in the target implementation processes  $n = 256$  coefficients per polynomial using eight iterative stages. These stages correspond to  $\log_2 n = 8$ .



**Fig. 4:**

NTT architecture in target implementation [\[34\]](#). The design features a  $2 \times 2$  butterfly architecture for optimized performance.

[Figure 4](#) illustrates the NTT architecture of the studied MLDSA module, a pipelined design optimized for processing four coefficients per cycle. The inputs to the NTT are four coefficients read from memory and four precomputed  $n$ -th root of unity ( $\omega$ ) stored in ROM. The architecture features a  $2 \times 2$  butterfly unit configuration with two stages of butterflies, each stage containing two butterfly units. This pipelined approach allows the initial stage to compute the first layer of the NTT transformation for four coefficients in each cycle while the second stage simultaneously processes the next layer. This design minimizes memory

overhead by eliminating the need to write intermediate results back to memory between layers.

The memory system in the NTT design stores input coefficients, intermediate results, and transformed values, with each memory address holding four coefficients. The memory bus width is defined as  $4 * \log_2(q - 1)$ , where  $q$  is the prime modulus used in the NTT. During operation, coefficients are read in a specific sequence and buffered to synchronize with the pipeline. A 4-to-1 multiplexer (MUX) routes the coefficients to the butterfly units, and after processing, the results are directed back to memory via a 2-to-1 MUX. The 2-to-1 MUX enables selection between initializing the memory with polynomials in the non-NTT domain or passing intermediate NTT values from the butterfly units to continue with the next stage.

This NTT architecture also facilitates hardware resource sharing across other ML-DSA operations, such as pointwise multiplication and pointwise addition and subtraction. Each butterfly unit consists of a modular adder, a modular subtractor, and a modular multiplier, all of which operate with the prime modulus  $q$ . Instead of utilizing dedicated multipliers, adders, and subtractors for pointwise operations, the open-source MLDSA implementation [34] design reuses the butterfly units within the NTT module to perform these tasks. This approach optimizes resource use but introduces shared vulnerabilities, as all operations rely on the same hardware.

## A. NTT in Target Implementation

### B. Pointwise Multiplication in Target Implementation

Pointwise multiplication in the target implementation performs element-wise multiplication of two polynomials in the NTT domain. As shown in Figure 5, the architecture reuses key NTT components, including memory, multiplexers, and pointwise multiplication cores instead of butterfly cores, to optimize resource usage while supporting the modular arithmetic required for cryptographic operations.



**Fig. 5:**

Pointwise multiplication architecture in target design [34]. This design reuses the hardware utilized for the NTT block.

In this implementation, two memories are used to store the input polynomials, with each memory address holding four coefficients. The cores execute four modular multiplications in parallel, enabling the processing of multiple coefficients concurrently. A multiplexer is used to direct the coefficients to the appropriate processing units [34].

The modular multiplication cores implement a reduction algorithm. Pointwise multiplication enables concurrent processing and ensures results remain within the finite field defined by  $q=8,380,417$  through modular arithmetic. Modular reduction maintains all coefficients within the range  $[0, q)$  after multiplying two 23-bit coefficients and obtaining a 46-bit product.

### C. Internal Structure of Modular Reduction

These modular multiplication cores use a unique reduction algorithm. It is specifically optimized for the prime modulus  $q=8,380,417$ , leveraging the modulus's advantageous structure to enhance throughput [34]. Unlike conventional reduction methods, this design implements a fully pipelined architecture that utilizes shifter and adder logic to perform modular reduction efficiently and with a constant execution time. The reduction circuit processes 46-bit intermediate products, which result from the multiplication of 23-bit coefficients during pointwise polynomial multiplication.

We explain the modular reduction process using the multiplication of  $\hat{s}_1$  the secret polynomial and  $\hat{c}$  the challenge polynomial as an example. These specific variables are chosen for illustration because they represent the focus of our sidechannel analysis, being the first point of interaction between the secret key

coefficients and deterministically generated, known values.

The modular reduction begins with the computation of the 46-bit intermediate product:

$$a = \hat{s}_1[i][j] \cdot \hat{c}[i]$$

► [View Source](#) ⓘ

where  $\hat{s}_1[i][j]$  is the  $j$ -th coefficient of the  $i$ -th polynomial in  $\hat{s}_1$ , and  $\hat{c}[i]$  is the  $i$ -th coefficient of  $\hat{c}$ . The modulus  $q$  is defined as:

$$q = 8,380,417 = 2^{23} - 2^{13} + 1,$$

► [View Source](#) ⓘ

which allows the reduction to leverage the equivalence:

$$2^{23} \equiv 2^{13} - 1 \pmod{q}.$$

► [View Source](#) ⓘ



**Fig. 6:**

The internal structure of the modular reduction circuit, as adapted from [34], is optimized for the prime modulus  $q$ . This design is particularly relevant for operations involving the multiplication of the NTT-transformed private polynomial  $\hat{s}_1$  and the NTT-transformed challenge polynomial  $\hat{c}$ , where  $\hat{s}_1[i][j]$  represents the  $j$ -th coefficient of the  $i$ -th polynomial in  $\hat{s}_1$ , and  $\hat{c}[i]$  is the  $i$ -th coefficient of  $\hat{c}$ .

The reduction algorithm decomposes the product  $a$  into bit segments:

$$a = 2^{13}a_{45:23} - a_{45:23} + a_{22:0},$$

► [View Source](#) ⓘ

where  $a_{45:23}$  and  $a_{22:0}$  represent the higher and lower bit ranges, respectively. To streamline computation, the reduction is defined as:

$$a = 2^{13}b - (a_{45:43} + a_{45:33} + a_{45:23}) + a_{12:0},$$

► [View Source](#) ⓘ

where:

$$b = a_{45:43} + a_{42:33} + a_{32:23} + a_{22:13}, \quad b < 2^{12}.$$

[▶ View Source](#) ⓘ

Further reduction of  $b$  is computed as:

$$2^{13}b = 2^{13}d - b_{11:10},$$

[▶ View Source](#) ⓘ

with:

$$d_{10:0} = b_{11:10} + b_{9:0}.$$

[▶ View Source](#) ⓘ

In the other stage, the 15 -bit value  $c_{14:0}$  is computed as:

$$c_{14:0} = a_{45:43} + a_{45:33}.$$

[▶ View Source](#) ⓘ

Additionally, the 23 -bit value  $e_{22:0}$  is computed as:

$$e_{22:0} = a_{45:23} + c_{14:0}.$$

[▶ View Source](#) ⓘ

Finally, the modular reduction result is:

$$a = d_{10:0} + a_{12:0} - e_{22:0},$$

[▶ View Source](#) ⓘ

where:

$$e_{22:0} = (a_{45:43} + a_{45:33} + a_{45:23}) + b_{11:10}.$$

[▶ View Source](#) ⓘ

## D. Exploiting Modular Reduction Registers

[Figure 6](#) exhibits the implementation details of the modular reduction circuit. This circuit features a three-stage pipelining process. In the first stage, the intermediate product  $a = \hat{s}_1[i][j] \cdot \hat{c}[i]$  is computed during pointwise multiplication and buffered with 46 -bit registers. In the second stage, the reduction circuit performs two modular addition operations in parallel, and these operations' intermediate values are stored in modular addition registers. Finally, the third stage performs a modular subtraction to ensure the result ranges within  $[0, q)$ . Throughout these stages, all intermediate values are stored in registers, forming

potential leakage points that can be exploited through side-channel analysis.

The attack methodology exploits the deterministic behavior of the target hardware design, as implemented in the deterministic version of the IP [34], to target specific registers. It is worth noting that the IP also supports a non-deterministic configuration, which may impact the applicability of this methodology. To conduct the attack, different input messages are provided to the ML-DSA implementation [34], which computes the corresponding challenge polynomial  $c$ . This polynomial then interacts with the secret key  $s_1$  through pointwise multiplication. Each coefficient of  $s_1$  and  $c$  is multiplied and subsequently processed by the reduction circuit, making the registers in this pipeline prime candidates for leakage exploitation.

The first vulnerability arises in the initial set of registers, which store the 46-bit intermediate product from the pointwise multiplication. By controlling the input message, the attacker can determine the exact value of the corresponding  $c$  coefficient. Using this knowledge, a hypothetical table for CPA is generated for each possible  $s_1$  coefficient. Since  $\hat{s}_1$  ranges between 0 and  $q - 1$  after NTT transform, the attacker generates  $q - 1$  hypothetical guesses. CPA is then performed as described in the II-D, correlating observed power traces with these hypothetical values. While this approach can reveal the coefficients of  $s_1$ , it might be prone to false positives due to the linear nature of multiplication [41].



**Fig. 7:**  
Experimental setup for capturing power traces.

To enhance the attack and eliminate these false positives, we use the next set of registers in the modular addition stage. These registers store intermediate values computed by the addition as part of the modular reduction process. The same  $s_1$  hypothetical guesses are reused, but the intermediate values in the CPA are adjusted to reflect the modular addition operation. By targeting this stage, the attack eliminates the false positives encountered from the first register set.

**Zeroization in Target Design.** A key aspect of this design is its zeroization method, implemented to prevent data recovery. Zeroization is a method of securely erasing cryptographic keys, critical security parameters, and electronically stored data by altering or clearing storage contents to prevent data recovery [42]. The RoT enforces the ML-DSA module in the cryptographic IP to perform zeroization after the key generation and signing processes. All internal registers are cleared using a software-triggered mechanism, where a singlecycle pulse on the hardware interface zeroes the first register before the next cryptographic operation begins. This ensures that the first pointwise multiplication core always starts with a clean state by removing residual data from previous operations and preventing the retention of sensitive information across cryptographic stages. While the zeroization mechanism in the studied ML-DSA module [34] is implemented as per FIPS 140-2 standards [42] to enhance security, it inadvertently aids side-channel analysis. By resetting all internal registers to zero before the start of key generation and signing operations, the mechanism simplifies attacks. The predictable initial state of the registers reduces the complexity of side-channel analysis, as attackers can leverage this knowledge to calculate the Hamming distance more easily and extract sensitive information. This dual effect highlights the importance of carefully balancing security measures to avoid unintentionally introducing vulnerabilities.



**Fig. 8:**  
Power traces of the ML-DSA signing operation. The top trace shows the full signing operation with multiple rejection loops. The middle trace isolates the NTT and pointwise multiplication stages, while the bottom trace zooms in on the initial pointwise multiplication.

In the Hamming Distance leakage model, power leakage correlates with the bitwise XOR of a register's current and previous states. To exploit this model, the attacker needs to know or guess the initial state of the register before making hypotheses about subsequent states. Without zeroization, the initial state of each register would be unknown, requiring the attacker to simultaneously guess both the initial state and the next state of the register. This significantly complicates the attack, as it doubles the size of the hypothetical guess table by introducing an additional dimension of uncertainty for each coefficient.

With zeroization, the registers are guaranteed to start from a known state of zero per captured execution. This eliminates the need to guess the initial state, reducing the guess table size by half and simplifying the analysis. The attacker can focus solely on guessing the next state of the registers based on hypothetical values of the secret coefficients  $s_i$ . This reduction in complexity allows the attack to be performed more efficiently, as the Hamming Distance model aligns with the deterministic transitions introduced by the zeroization process.

## SECTION V. Evaluation Result

This section provides an overview of the experimental setup used to evaluate the proposed side-channel attack, including the hardware and software configuration for capturing and analyzing power traces. It also presents the results of leakage detection and correlation analysis, demonstrating the effectiveness of the attack in exploiting vulnerabilities in the ML-DSA hardware implementation.



**Fig. 9:**

Results of the proposed side-channel analysis targeting two registers in the hardware implementation of ML-DSA. The first row shows the attack results for the multiplication result register, including the correlation plot (a), Pearson correlation evolution (b), and the rank of the correct key, which reaches the first position (c). The second row presents similar results for the modular arithmetic operation register, highlighting the correlation plot (d), Pearson correlation evolution (e), and the rank of the correct key (f).

### A. Experimental Setup

The power traces of the signing execution in ML-DSA were collected using the ChipWhisperer-Lite and the CW305 Artix7 FPGA target board [43] for side-channel evaluation. The experimental setup, shown in [Figure 7](#), includes a Low-Pass Filter, a PicoScope 6404E oscilloscope [44], and the CW305 FPGA development board, which is equipped with a Xilinx Artix-7 chip. The FPGA was programmed with the opensource ML-DSA implementation <sup>1</sup>, an implementation of MLDSA [34], configured to operate at security level 5.

The oscilloscope samples power traces at a frequency of 156.25 MHz, corresponding to a sampling interval of 6.4 ns. Two probes are connected to the victim device: Channel A records the power traces with a noise filter applied, while Channel B captures the trigger signal. The FPGA board interfaces with the ChipWhisperer capture board for data acquisition and synchronization. The CPA analysis was conducted on a computer equipped with an Intel(R) Xeon(R) E5-1620 CPU running at 3.50 GHz and 32 GB of memory. The CPA program was implemented in Python 3.10.

[Figure 8](#) illustrates the power traces captured during the ML-DSA signing process. The top subplot shows the complete signing operation, including multiple iterations of the rejection loop caused by out-of-bound signature values. The middle subplot focuses on the computational stages, including the INTT and pointwise multiplication. The bottom subplot zooms in on the pointwise multiplication step, showing the power transitions recorded during this critical operation. These power traces form the basis for side-channel

evaluation in our experimental setup.

## B. Visualizing Leakage: Evolution of Correlation

To further investigate the vulnerability of the target implementation, we conducted a CPA attack. The results of the sidechannel analysis are presented in [Figure 9](#), which visualizes the correlation evolution, Pearson correlation coefficients, and the rank of the correct key for two critical attack points. These figures provide a comprehensive overview of the leakage patterns and highlight the effectiveness of the proposed CPA methodology in exploiting these vulnerabilities.

The first attack targets the register holding the 46-bit intermediate result of the multiplication between the 23-bit coefficients of the secret polynomial vector  $s_1[i]$  and the challenge polynomial  $\hat{c}[i]$ . As depicted in [Figure 9a](#), the correlation plot illustrates the leakage caused by state transitions in this register. The attack leverages the Hamming Distance leakage model to correlate the power consumption with transitions in the register, enabling the adversary to distinguish the correct key from incorrect guesses. After processing 10,000 traces, the correlation associated with the correct key surpasses all other candidates. This observation is further supported by the Pearson correlation evolution in [Figure 9b](#), which shows the correct key consistently demonstrating a progressively stronger and statistically significant correlation (99.99 %) compared to incorrect guesses. The rank of the correct key, shown in [Figure 9c](#), converges rapidly, demonstrating the success of the attack in recovering the secret polynomial coefficients. The effectiveness of this attack is attributed to the predictable nature of the pointwise multiplication operation and the leakage arising from state transitions within the register storing the multiplication result.

The analysis was extended to a second attack point: the modular addition operation within the reduction circuit to evaluate the generality and robustness of the CPA methodology. This stage processes the intermediate multiplication result through modular arithmetic, temporarily storing intermediate values in registers. The internal structure of the modular reduction submodule, as illustrated in [Figure 6](#), consists of modular addition and subtraction operations that follow the same flow and structure. These consistent computational patterns introduce predictable leakage, making the modular addition registers an equally viable target for side-channel analysis. [Figure 9d](#) demonstrates the correlation plot for this attack point, where the correct key is clearly distinguishable from other guesses. Similarly, the Pearson correlation evolution in [Figure 9e](#) demonstrates that the correct key progressively achieves higher and statistically significant correlation values (99.99 %) compared to incorrect guesses as the number of traces increases. The rank of the correct key, presented in [Figure 9f](#), converges rapidly, further validating the success of this attack point.

The effectiveness of these attacks is further facilitated by the constrained key guessing space in the hardware implementation [34], where modular reduction is applied after each butterfly operation during the NTT computation. Specifically, the key space in hardware is restricted to  $[0, q)$ , which is significantly smaller compared to the broader key range in the reference software implementation of ML-DSA [45], spanning  $[-\eta - 8(q - 1), \eta + 8(q - 1)]$ . This smaller key space reduces the complexity of the attack, making it easier to pinpoint the correct key.

The results from both attack points conclusively demonstrate that the open-source implementation of ML-DSA [34], an implementation of ML-DSA, remains vulnerable to sidechannel leakage, particularly in the modular arithmetic stages. The CPA methodology employed in this paper effectively exploits predictable state transitions and the repetitive nature of modular arithmetic operations, emphasizing the necessity of robust countermeasures in hardware implementations of postquantum cryptographic algorithms.

## SECTION VI. Discussion



Our attack uncovers critical vulnerabilities in the targeted cryptographic implementation [34], specifically targeting the NTT-based polynomial pointwise multiplication. At the time of this study, the implementation lacked side-channel countermeasures such as masking [27]– [30] or shuffling [31], leaving it highly susceptible to side-channel leakage. While our findings highlight these vulnerabilities, the evaluation and implementation of countermeasures fall outside the scope of this work. Instead, our primary objective is to provide a detailed analysis of the identified weaknesses to inform the development of more secure cryptographic designs.

This work focuses on pinpointing vulnerabilities in the ML-DSA hardware implementation [34], particularly those arising in the NTT-based pointwise multiplication. This study emphasizes the critical need to address potential vulnerabilities in hardware implementations to prevent exploitation. By identifying these risks and communicating them to the broader cryptographic and hardware security communities, we aim to encourage proactive measures to mitigate side-channel attacks before they can be leveraged in practice.

It is important to note that the vulnerabilities identified in this work may not be confined to the NTT-based pointwise multiplication. Other components of the design [34], such as encoding, decoding, and NTT, could also leak sensitive information. Future research should explore these areas to ensure comprehensive protection against side-channel attacks and uncover additional weaknesses that may exist in the design.

Additionally, advanced techniques such as machine learning-based side-channel analysis [13]– [20] present a promising avenue for future research. These methods can identify complex or previously hidden leakage patterns, offering deeper insights into hardware vulnerabilities and informing the development of more robust countermeasures. Machine learning-based approaches also enable more generalized evaluations across different cryptographic implementations, enhancing their utility in securing hardware designs.

This study underscores the importance of systematically analyzing and addressing side-channel vulnerabilities in cryptographic implementations. As cryptographic algorithms transition from theoretical constructs to practical hardware deployments, ensuring resilience against both mathematical and physical attacks becomes imperative. By exposing these vulnerabilities and offering directions for future research, this work seeks to bridge the gap between theoretical security and practical robustness, advancing the security of hardware cryptographic designs in real-world applications.

## SECTION VII.

# Conclusion

Authors	This paper exposes vulnerabilities in the hardware implementation of ML-DSA [34], specifically within an open-source Silicon RoT internal block. Using CPA, we demonstrated that critical cryptographic operations-	▼
Figures	modular reduction during NTT-based polynomial pointwise multiplication-are vulnerable to side-channel leakage. Our analysis shows that an adversary can extract secret keys with as few as 10,000 power traces,	▼
References	posing a significant threat to firmware verification and RoT integrity. These findings underscore the need for robust countermeasures to secure post-quantum cryptographic implementations and highlight the	▼
Citations	importance of addressing advanced attack methodologies to mitigate emerging risks.	▼
Keywords		▼
ACKNOWLEDGMENT		
Metrics	This work is supported in part by NSF-2147196.	▼
Footnotes		▼

IEEE Personal Account

CHANGE USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS

VIEW PURCHASED DOCUMENTS

Profile Information

COMMUNICATIONS PREFERENCES

PROFESSION AND EDUCATION

TECHNICAL INTERESTS






Need Help?


US & CANADA: +1 800 678 4333

WORLDWIDE: +1 732 981 0060

CONTACT & SUPPORT

Follow

About IEEE Xplore | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting  | Sitemap | IEEE Privacy Policy

A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2026 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.