

Designing Post-Quantum Cryptography for Resilience against AI-Assisted Side-Channel Attacks

Abubakr Abdulgadir, Luke Beckwith, Reza Azarderakhsh
PQSecure Technologies LLC
Boca Raton, FL, USA
{abubakr.abdulgadir, luke.beckwith, razarder}@pqsecurity.com

Abstract—Development and integration of post-quantum cryptographic implementations is accelerating due to government mandates motivated by quantum computing progress. Profiled attacks including those assisted by AI, threaten real-world deployments of these cryptographic designs. In this work, we discuss methods to harden devices, including traditional side-channel countermeasures as well as a novel training phase countermeasure based on making seeds used for countermeasures unpredictable and uncontrollable by adversaries. We also present experimental results showing the degradation of observable leakage given that countermeasures are implemented and enforced by the hardware.

Index Terms—AI-Assisted Attacks, Post-Quantum Cryptography, Hardware Accelerators

I. INTRODUCTION

Deployments of public key cryptography are being actively upgraded to post-quantum cryptography (PQC). Advances in quantum computing threaten classical public key cryptography motivating governments to mandate the transition to PQC. Standards for key encapsulation and digital signatures such as ML-KEM and ML-DSA [5, 4] are already implemented in hardware and software including side-channel protected designs.

For applications where adversaries can gain access to cryptographic devices, side-channel analysis can be used to extract secret information which threaten successful deployments.

AI-assisted attacks typically fall into the category of profiled side-channel attacks. In this setting, an attacker first performs a training phase, during which a machine-learning model is trained using traces collected from a device operating with known inputs and known secret keys. The goal of this phase is to learn the statistical relationship between observed side-channel leakage and the internal sensitive variables of a cryptographic algorithm. During the attack phase, the trained model is applied to traces collected from the device under attack, where the secret key is unknown. Using the patterns learned during training, the model predicts the most likely value of the sensitive information. The training phase generally requires a large number of traces, whereas the attack phase often requires only a small number of traces—sometimes even a single trace—to recover candidate key information.

Template attacks [2] represent a classic example of profiled side-channel attacks. In template attacks, the training phase is used to construct statistical models, typically multivariate Gaussian distributions, that characterize the leakage corresponding to different sensitive values. During the attack phase, the observed leakage is compared against these models to determine the most likely key hypothesis.

Compared with traditional methods, AI-assisted attacks offer several advantages. In particular, they often require little or no manual preprocessing of traces and can automatically

learn complex and subtle correlations between side-channel measurements and sensitive internal variables that cannot be easily captured in simple statistical models. An AI-assisted side-channel attack uses machine learning, most often deep learning, to infer secret-dependent information from leakage. Instead of relying solely on classical leakage models such as Hamming weight. This is especially useful when the target implementation uses protections such as masking or shuffling, because neural networks can sometimes recover hidden structure that is difficult to model analytically. Figure 1 illustrates the general pipeline of a single-trace AI-assisted side-channel attack, where traces are collected, processed, used to train a model, and then exploited to recover secret information.

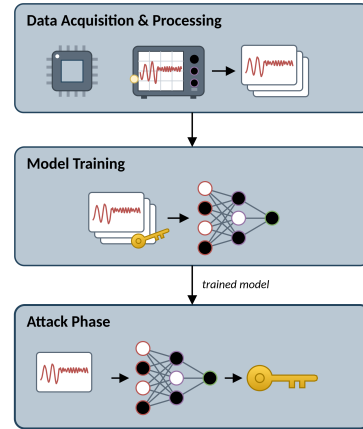


Fig. 1: Single trace AI-assisted attack overview

II. AI-ASSISTED ATTACKS ON PQC IMPLEMENTATION

AI and machine-learning-based attacks on cryptographic algorithms are surveyed in [8] and [7], while classical side-channel attacks on post-quantum cryptographic schemes are discussed in [11]. In the following, we review the state of the art in AI-assisted side-channel attacks on post-quantum cryptography.

PQC implementations contain structured operations that repeatedly manipulate secret-dependent values. These operations create consistent leakage points that AI-based side-channel attacks can exploit. Figure 2 shows the side-channel-sensitive operations for the ML-KEM and ML-DSA standards. These operations include polynomial arithmetic operations such as NTT and sampling among others.

A. Attacks on Key Encapsulation Mechanisms

Before discussing scheme-specific attacks, it is helpful to understand the general workflow of a key encapsulation

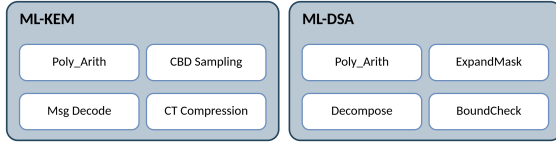


Fig. 2: ML-KEM and ML-DSA side-channel-sensitive operations

mechanism (KEM), shown in Figure 3. In a typical PQC KEM protocol, Alice first performs key generation to produce a public key and a secret key. The public key is then sent to Bob. Using this public key, Bob performs encapsulation to generate a ciphertext and a shared secret. The ciphertext is sent back to Alice, who runs decapsulation using her secret key to recover the shared secret. As illustrated in Figure 3, the most sensitive operations occur during key generation and decapsulation, since these steps directly process the secret key. Because of this, these stages become primary targets for side-channel analysis. Many recent AI-assisted side-channel attacks exploit leakage from these operations to recover secret-dependent information in post-quantum cryptographic schemes such as ML-KEM and HQC.

ML-KEM In [3], the authors examined a masked and shuffled software implementation of Kyber and showed that AI-assisted side-channel analysis can be strengthened when combined with fault injection. Their attack first uses a voltage glitch to skip the shuffling countermeasure, thereby reducing the effectiveness of the implemented side-channel protections. After bypassing shuffling, the authors apply deep learning-based power analysis to the masked message-encoding step during the decapsulation process. By exploiting the resulting leakage, the attack is able to recover the encapsulated message, which in turn allows recovery of the shared secret key produced during decapsulation. The results show that the combination of fault injection and neural-network-based side-channel analysis can transform a protected software implementation into a single-trace attack target.

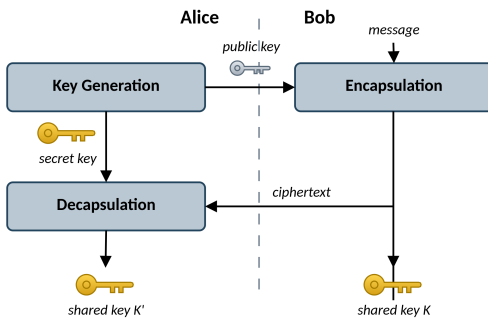


Fig. 3: Generic PQC KEM workflow showing key generation, encapsulation, and decapsulation.

In [10], the authors introduce a blind side-channel attack on Kyber that targets the pointwise multiplication operation to fully recover the secret key. Unlike many conventional side-channel attacks, blind attacks do not require knowledge of the processed ciphertext, which makes them feasible even when the attacker has no access to the communication channel. The proposed attack employs a machine-learning-based classifier

to exploit leakage from a software implementation running on an ARM Cortex-M4 processor, demonstrating that sensitive operations in Kyber can still be vulnerable to practical AI-assisted side-channel analysis.

B. Attacks on Digital Signature Algorithms

Before discussing attacks on ML-DSA, it is useful to understand the general workflow of a post-quantum digital signature scheme, illustrated in Figure 4. In this process, Alice first performs key generation to produce a public key and a secret key. The public key is shared with Bob. When Alice wants to send a message, she uses the secret key to perform signature generation, producing a signature associated with the message. Bob then performs verification using the public key to check whether the signature is valid. If the verification succeeds, the message is accepted; otherwise, it is rejected. As shown in Figure 4, the most sensitive computations occur during key generation and signature generation, since these steps directly use the secret key. Because of this, they become primary targets for side-channel analysis. Many recent side-channel attacks exploit leakage from these operations to recover secret-dependent information in post-quantum signature schemes such as ML-DSA and Falcon.

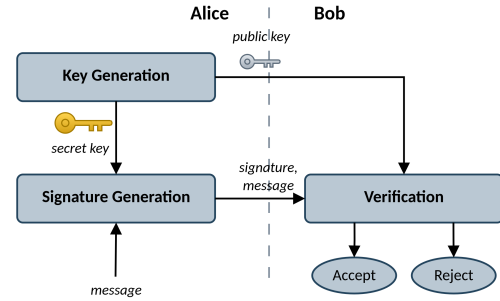


Fig. 4: Generic workflow of a post-quantum digital signature scheme.

ML-DSA In [13], the authors identify secret-key unpacking during signing as a previously overlooked leakage point in Dilithium/ML-DSA, rather than the commonly targeted multiplication operation involving $c \cdot s_1$. Because the unpacking routine processes the coefficients of the secret vectors s_1 and s_2 , it directly leaks secret-dependent information. The authors exploit this leakage using a profiled deep-learning-based power analysis attack, demonstrating that half of the coefficients of the secret vectors can be recovered from a single power trace. Since the leakage reveals only part of the secret key, recovering the entire secret key requires an additional recovery step. The authors proposed two recovery methods. If the low bits t_0 of the public key are known, the remaining coefficients can be recovered by solving a system of linear equations derived from the Dilithium key-generation relations. If t_0 is not available, a larger subset of the s_1 coefficients is first recovered from side-channel leakage, and the remaining coefficients are reconstructed using lattice reduction techniques. The authors evaluate their attack on a Cortex-M4 implementation. To resist this attack, the authors discussed feeding the secret coefficients in shares and shuffling their order.

In [9], the authors present a practical side-channel attack on Dilithium that targets the signing relation $z = y + c \cdot s_1$.

The attack combines two profiling approaches. First, linear regression-based profiling is used to recover information about the ephemeral vector y . Second, a convolutional neural network (CNN)-based profiling method is applied to recover information about the product $c \cdot s_1$. The information obtained from these two approaches is then combined to strengthen the overall key-recovery attack.

III. HARDENING HARDWARE PQC IMPLEMENTATIONS

Countermeasures that introduce noise and uncertainty into side-channel traces are useful against profiled attacks, including AI-assisted side-channel attacks. Techniques such as masking and shuffling are particularly useful during the attack phase, as they complicate the attacker's ability to associate observed leakage with secret-dependent computations directly. However, recent work has shown that these countermeasures can sometimes be weakened or bypassed. For example, in [3], the authors demonstrate that the shuffling countermeasure can be disabled via a fault-injection attack, leaving the implementation vulnerable to AI-assisted key-recovery attacks. Additionally, when training AI models, it is helpful for the attacker to use a device with disabled countermeasures. For example, in the attack in [6], 80% of the traces used to train the model were obtained from a device with known shuffling order. Thus it is not sufficient to only apply attack-phase countermeasures. In this section, we discuss not only attack-focused countermeasures, but also a training-focused countermeasure which reduces the attacker's ability to control the random seeds used to initialize masking and shuffling.

a) Attack Countermeasures: In dedicated hardware (FPGA/ASIC), we can manipulate the Signal-to-Noise Ratio (SNR) to resist AI learning. Parallelism can produce noise where high amount of switching activity that is not related to the secret value takes place at the same time the secret value is processed. As a result, the power consumption of a secret-dependent logic gate is hidden under the switching noise of all other parallel components. For an AI model to learn patterns, it must observe the tiny fluctuations of a secret gate among the noise of other gates. By increasing the complexity of parallel operations, we effectively hide the signal in noise. This makes attacks, especially single-trace attacks, more challenging, as the model struggles to find the patterns it needs to learn.

Masking is a countermeasure in which a secret value k is decomposed into n randomized shares, for example, ($k = k_1 + k_2 + k_3$) in an additive sharing scheme. The shares are constructed so that any subset of fewer than all required shares reveals no information about the underlying secret. Consequently, an attacker must recover and combine the leakage associated with all shares before mounting a successful attack.

Shuffling is an effective technique for further reducing the signal-to-noise ratio of a side-channel implementation. More specifically, shuffling spreads the leakage associated with a sensitive computation over a wider time interval compared with an unshuffled design. This temporal spreading makes the leakage less concentrated and therefore harder to exploit. It is particularly effective against AI-based attacks, since many models, including convolutional neural networks (CNNs), rely on stable local patterns appearing at consistent time offsets across traces. Shuffling disrupts this structure by fragmenting these patterns and placing their components in a random order

from one execution to another, which significantly complicates the learning task.

Consequently, an attacker must first recover or approximate the hidden alignment before the traces can be meaningfully exploited. This additional burden can substantially reduce the effectiveness of AI-assisted side-channel attacks, especially when shuffling is combined with other countermeasures that further suppress leakage. In some cases, the attacker may have find the correct order of recovered shuffled data. For example, even if an attacker recovered each of the 256 coefficients of an ML-KEM shuffled polynomial in a single trace, the attacker will need significant computation power to reorder the coefficients.

b) Training Countermeasures: In the training phase, the attacker collects a large number of traces with changing input from a clone of the target device. The training data set is then composed from the traces as input features and intermediate values as labels. This dataset is then used to train an AI model that learns the relationship between the power traces and the sensitive intermediate values. In the attack phase, one or a small number of traces are collected from the target device and fed to the model to predict the sensitive information. The success of this method depends heavily on the quality of the training dataset. Specifically, repeatable and temporally aligned traces where operations run at the same time relative to the start of the execution are attacker-friendly.

Masking and shuffling together significantly reduces leakage from cryptographic devices. However, if an attacker can control the seeds that are used to generate randomness for these countermeasures during profiling, the protection can be weakened. For example, by fixing the seeds to a fixed value, the attacker can force the same masks and shuffling order across different cryptographic calls. This helps the attacker to collect consistent traces that allow AI models to be trained efficiently.

To address this vulnerability, we propose to remove the ability to control or predict the countermeasure seeds by generating them internally and never accepting them from software. Specifically, we propose using a seed derivation function (SDF) to generate the seed from a combination of input data, internal randomness, optional external randomness, and a device-resident secret K_{device} . The seed is derived as follows:

$$S_{sca} = SDF(r_{internal}, ctr, K_{device}, r_{external}, input_data)$$

Where $r_{internal}$ is internally generated randomness, for example, from an SRAM startup pattern. K_{device} is a device-resident secret key that could be stored, for example, in eFuse or One-time Programmable memory (OTP). This construction avoids the usage of a TRNG, which may be costly. Additionally, a TRNG can be shared at the system level with other components, with software handling reading randomness from the TRNG and providing it to the cryptographic core. In this case, an attacker with full software control can provide compromised seeds to the cryptographic core. The derivation function may be implemented using a hash function or XOF, such as SHAKE, which are already part of many post-quantum algorithms. Figure 5 depicts the proposed architecture.

The main objective of this approach is to prevent the attacker from fixing or predicting the seed used to generate the random bits used by the shuffling and masking countermeasures. Even if the attacker has full control over the software

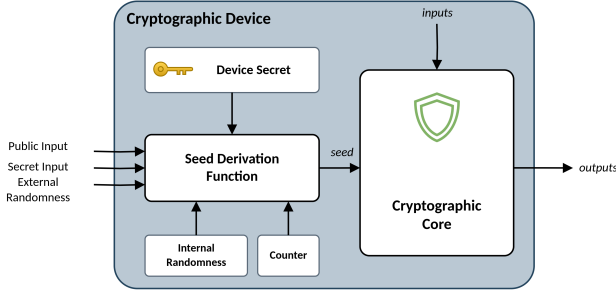


Fig. 5: Seed derivation function overview

and can supply all inputs, including $r_{external}$, the final countermeasure seed remains unknown because it depends on K_{device} and is unfixable because of its dependence on the input data, $r_{internal}$ and the counter value. For profiling, the attacker needs to vary the input; however, in this construction, doing so forces the seed to change, causing randomization of the shuffling order and masks. The end result is that the attacker no longer obtains a stable and regular leakage pattern for sensitive operations. Instead, the leakage temporal order and underlying intermediate share values are randomized across calls to the cryptographic function.

Using a device key removes dependence on implementation obscurity. Even if the adversary knows the seed derivation function and the device architecture, they still need the device secret to predict the seed. Additionally, the inclusion of $r_{internal}$ and a counter prevents the attacker from fixing the seed and averaging across multiple calls with the same input. Accepting the external randomness $r_{external}$ allows legitimate users to strengthen the entropy of the seed generation input. At the same time, an attacker cannot use $r_{external}$ to control seeds. Additionally, we cannot depend on $r_{internal}$ alone for the seed since it may be fresh only after the device boots. This is indeed the case when using SRAM startup patterns. The counter is updated between invocations of the cryptographic module to avoid seed repetition. Both $r_{internal}$ and the counter initialization occur only on power-on reset.

A side-channel attack may be attempted on the SDF to recover K_{device} . Since the same K_{device} is used, it is a differential power analysis (DPA) target. To resist this attack, we propose to absorb the internal randomness first and then K_{device} into the SDF state and finally accept data from the user. This way, any intermediate value will depend on many K_{device} bits, making key guessing challenging. To resist simple power analysis (SPA) and profiled attacks on the SDF, the SDF implementation can use shuffling and a wide datapath, for example, at least 64 bits.

An attacker may attempt to bypass this countermeasure using a fault injection; however, this is significantly more difficult in the profiling phase than in the attack phase. The profiling phase usually requires thousands or millions of traces for training. Hence, the attacker needs a reliable and repeatable fault for every collected trace. This repeatable fault is more difficult to achieve and easier to defend against. For example, randomly delaying the call for the seed derivation function and using a dynamic counter can respectively thwart or detect many fault injection attempts.

IV. CASE STUDY

In this section we present a case study for the attack phase countermeasures. A variety of methodologies exist for evaluating side-channel resistance. Among them, Test Vector Leakage Assessment (TVLA) is widely adopted in both academia and industry. In addition, the signal-to-noise ratio (SNR) serves as a quantitative metric to evaluate the extent to which a sensitive intermediate variable contributes to observable leakage relative to unrelated noise sources.

To assess the effectiveness of the shuffling countermeasure, we conducted an SNR-based leakage analysis focusing on two critical intermediate computations that have previously been exploited in AI-assisted attacks:

- ML-KEM decapsulation pointwise multiplication, and
- ML-KEM decapsulation plaintext message decoding.

The first attack vector enables full recovery of the private key, while the second allows recovery of the shared secret; both outcomes critically compromise the security of the cryptographic system.

A. Target Implementation

Here we briefly introduce the implementation used for the case study. As the focus of this paper is on high-level protection techniques, we will not discuss the details of the architecture. The platform is a HW/SW codesign system composed of a small VexRiscv processor [12] paired with a set of hardware coprocessors. The coprocessors implement all computations of the ML-DSA and ML-KEM algorithms with only the control logic being handled in software. There are four coprocessors employed in the design: one vector coprocessor processing polynomial computations, a masked hash coprocessor, a share converter coprocessor for converting between the boolean shares used for hashing and the arithmetic shares used for polynomial computations, and finally a PRNG coprocessor which generates the shuffling permutation and generates fresh randomness for masking gadgets. The coprocessors have no internal memories and are all connected to the CPUs data memory through an AXI interconnect. All secret key related computations are protected using masking and shuffling.

B. Experimental Setup and Dataset Collection

The cryptoprocessor was implemented on an AMD Artix-7 FPGA on a ChipWhisperer CW305 target board. The device under test (DUT) was clocked at 10 MHz. Power traces were acquired using a PicoScope 3000 oscilloscope at a sampling rate of 62.5 MS/s. A total of 50,000 traces were collected by executing ML-KEM decapsulation with random, valid test vectors. Two datasets were extracted from these traces:

- A segment corresponding to the execution of the pointwise multiplication $s \cdot u$.
- A segment corresponding to the plaintext message decoding stage.

Each dataset also includes the associated input data used during decapsulation. Representative traces are shown in Figures 6 and 7.

C. Test Procedure

Each trace was assigned a class label based on the targeted intermediate variable:

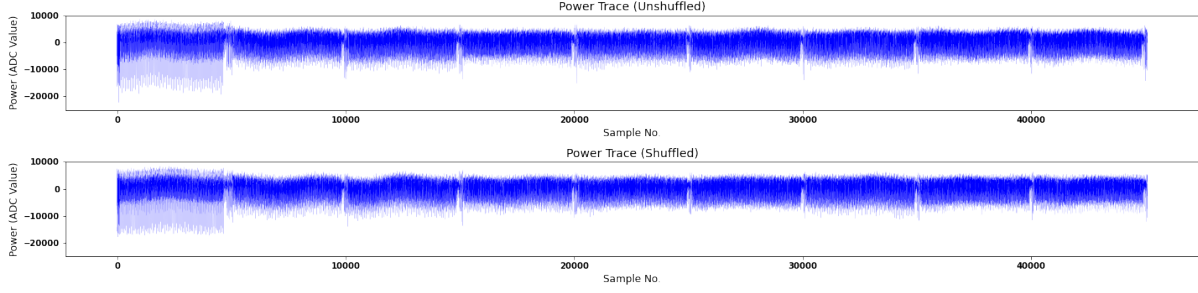


Fig. 6: Sample power trace while performing the multiplication $s \cdot u$. Top: unshuffled implementation, Bottom: shuffled implementation

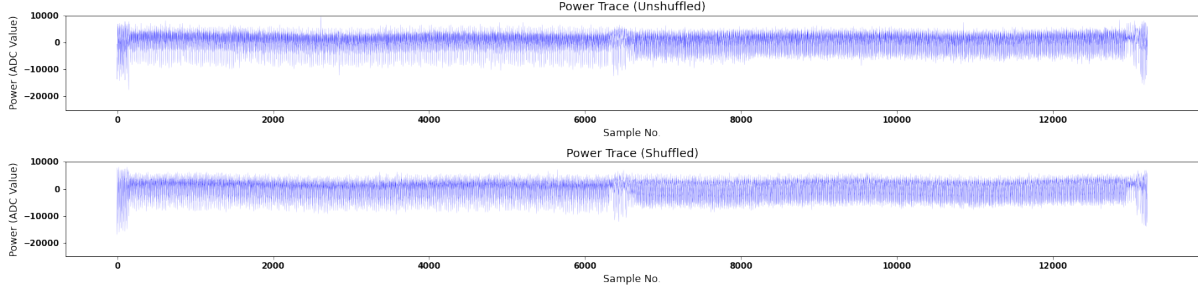


Fig. 7: Sample power trace while performing the plaintext decoding. Top: unshuffled implementation, Bottom: shuffled implementation

- For the pointwise multiplication experiment, traces were grouped according to the Hamming weight of a coefficient of the secret polynomial s .
- For the message decoding experiment, traces were classified based on the value of the corresponding message bit.

The labeled traces were then processed using the SNR computation routines provided by the SCALib library [1].

D. Results

a) Pointwise multiplication leakage: The first experiment targets the ML-KEM pointwise multiplication $s \cdot u$, where s is part of the private key. This operation has been previously exploited in AI-assisted attacks, resulting in full key recovery [10].

Figure 8 shows the SNR as a function of time for the first sixteen coefficients of s . The unshuffled design exhibits multiple significant leakage spikes across coefficients, whereas the shuffled implementation shows no distinguishable peaks. We also notice that odd coefficients have much less SNR, which is not observable in most cases, even in the unshuffled implementation. The pointwise multiplication of two polynomial vectors, a and b , is implemented using three inputs: the polynomial vector a , the polynomial vector b , and a cached polynomial vector, b_{cache} . The b_{cache} is precomputed in a prior step by multiplying the odd coefficients of b with their corresponding zeta values, allowing for more efficient computation during the multiplication process. In this operation, the secret key is provided as the second operand (b). As a result, modifying an even coefficient of the secret key affects a single entry in the pointwise multiplication, whereas modifying an odd coefficient impacts two entries. Consequently, changes to

odd coefficients introduce more noise into the computation compared to changes to even coefficients. We hypothesize that this added noise is the cause of low SNR for odd coefficients.

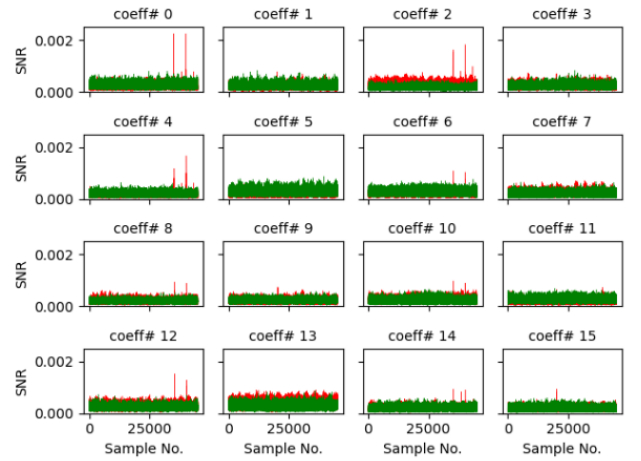


Fig. 8: SNR of the first 16 coefficients of the secret polynomial s . Red: unshuffled implementation, Green: shuffled implementation

Quantitatively, the maximum SNR in the shuffled design remains below 10^{-3} , while the unshuffled design reaches approximately 2.2×10^{-3} . Furthermore, the absence of prominent peaks in the shuffled case indicates that leakage is temporally dispersed and lacks strong distinguishing features.

b) Message decoding: The second experiment focuses on plaintext message decoding, performed at the end of ML-

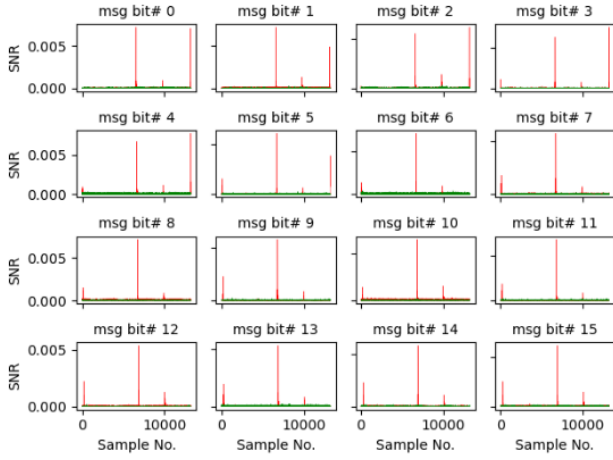


Fig. 9: SNR of the first 16 bits of the plaintext message. Red: unshuffled implementation, Green: shuffled implementation

KEM.PKE_Decrypt. Leakage at this stage directly enables recovery of the shared secret. This operation has also been targeted in prior AI-assisted attacks [3].

Traces were segmented and classified based on the value of the decoded message bit. SNR analysis was performed on the first 16 least significant bits (LSBs) of the message.

Figure 9 presents the SNR across the first 16 bits of the plaintext message. The unshuffled implementation exhibits clear leakage peaks aligned with message processing, whereas the shuffled implementation shows no observable peaks. The SNR remains uniformly low, indicating significant attenuation of exploitable leakage.

c) *Discussion:* Across both experiments, the shuffled implementation demonstrates a clear flattening of the SNR profile, with no distinct time points exhibiting elevated leakage. This indicates that the signal component associated with sensitive variables is effectively hidden by noise and temporal dispersion. As a result, both classical statistical attacks and AI-based methods will face significantly increased difficulty in extracting meaningful features required for key recovery. The absence of stable, high-SNR regions further reduces the ability of machine learning models to identify and learn leakage patterns.

V. CONCLUSIONS

In this paper we proposed a method that targets hardening hardware PQC implementations. The main idea is to resist attempts to profile a hardware device and train AI models even if the adversary have full access to the interface and controls input data including seeds used for countermeasures. We show that depriving the adversary of control and knowledge of the countermeasure seed makes collecting a high-quality dataset for training a challenging task. Additionally, we experimentally showed the significant degradation on SNR when shuffling is enforced in a hardware implementation ML-KEM algorithm.

INTELLECTUAL PROPERTY DISCLOSURE

Some of these techniques may be covered by US and/or international patents developed by PQSecure Technologies LLC.

REFERENCES

- [1] Gaëtan Cassiers and Olivier Bronchain. “SCALib: A Side-Channel Analysis Library”. In: *Journal of Open Source Software* 8.86 (2023), p. 5196. DOI: 10.21105/joss.05196.
- [2] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template Attacks”. In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, çetin K. Koç, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28. ISBN: 978-3-540-36400-9.
- [3] Sonke Jendral et al. “A Single-Trace Message Recovery Attack on a Masked and Shuffled Implementation of CRYSTALS-Kyber”. In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2024. DOI: 10.1109/HOST55342.2024.10545390.
- [4] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. Tech. rep. FIPS 204. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.204. URL: <https://doi.org/10.6028/NIST.FIPS.204>.
- [5] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Tech. rep. FIPS 203. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.203. URL: <https://doi.org/10.6028/NIST.FIPS.203>.
- [6] Kalle Ngo, Elena Dubrova, and Thomas Johansson. “A Side-Channel Attack on a Masked and Shuffled Software Implementation of Saber”. In: *Journal of Cryptographic Engineering* (Apr. 25, 2023). ISSN: 2190-8516. DOI: 10.1007/s13389-023-00315-3. URL: <https://doi.org/10.1007/s13389-023-00315-3> (visited on 06/07/2023).
- [7] Abiodun Olaluwe et al. *Machine Learning and Side-Channel Attacks on Post-Quantum Cryptography*. Cryptology ePrint Archive, Paper 2025/1754. 2025. URL: <https://eprint.iacr.org/2025/1754>.
- [8] Stjepan Picek et al. “SoK: Deep Learning-based Physical Side-channel Analysis”. In: *ACM Computing Surveys* 55.11 (Feb. 2023), 227:1–227:35. DOI: 10.1145/3569577.
- [9] Zehua Qiao et al. *Single Trace is All It Takes: Efficient Side-channel Attack on Dilithium*. Cryptology ePrint Archive, Paper 2024/512. 2024. URL: <https://eprint.iacr.org/2024/512>.
- [10] Prasanna Ravi et al. *Machine Learning based Blind Side-Channel Attacks on PQC-based KEMs — A Case Study of Kyber KEM*. Cryptology ePrint Archive, Report 2024/169. 2024. URL: <https://eprint.iacr.org/2024/169>.
- [11] Prasanna Ravi et al. “Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results”. In: *ACM Transactions on Embedded Computing Systems* 23.2 (2024), 35:1–35:54.
- [12] SpinalHDL. *VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation*. <https://github.com/SpinalHDL/VexRiscv>. 2026.
- [13] Ruize Wang et al. “Unpacking Needs Protection: A Single-Trace Secret Key Recovery Attack on Dilithium”. In: *IACR Communications in Cryptology* 1.3 (2024), p. 12.