Single-Trace Side-Channel Attacks on ω-Small Polynomial Sampling

with Applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM

Emre Karabulut[†], Erdem Alkim[‡], Aydin Aysu[†]
[†]Department of Electrical and Computer Engineering, North Carolina State University, NC, USA
[‡]Department of Computer Engineering, Ondokuz Mayis University, Samsun, Turkey

***Abstract*—This paper proposes a new *single-trace* side-channel attack on lattice-based post-quantum protocols. We target the ω-small polynomial sampling of NTRU, NTRU Prime, and CRYSTALS-DILITHIUM algorithm implementations (which are NIST Round-3 finalist and alternative candidates), and we demonstrate the vulnerabilities of their sub-routines to a power-based side-channel attack. Specifically, we reveal that the sorting implementation in NTRU/NTRU Prime and the shuffling in CRYSTALS-DILITHIUM's ω-small polynomial sampling process leaks information about the '-1', '0', or '+1' assignments made to the coefficients. We further demonstrate that these assignments can be found within a single power measurement and that revealing them allows secret and session key recovery for NTRU/NTRU Prime, while reducing the challenge polynomial's entropy for CRYSTALS-DILITHIUM. We execute our proposed attacks on an ARM Cortex-M4 microcontroller running the reference software submissions from NIST Round-3 software packages. The results show that our attacks can extract coefficients with a success rate of 99.78% for NTRU and NTRU Prime, reducing the search space to 2^{41} or below. For CRYSTALS-DILITHIUM, our attack recovers the coefficients' signs with over 99.99% success, reducing rejected challenge polynomials' entropy between 39 to 60 bits. Our work informs the proposers about the single-trace vulnerabilities of their software and urges them to develop single-trace resilient software for low-cost microcontrollers.**

***Index Terms*—Side-channel attacks, Post-quantum cryptography, NTRU, CRYSTALS-DILITHIUM**

I. Introduction

Quantum computers are proven to solve the discrete logarithm [1] and integer factorization [2] problems with exponential speedup [3]. Unfortunately, classical digital signature and public-key encryption schemes such as RSA [4] and elliptic curve cryptosystems [5] rely on these problems, which motivates the need for alternatives. To address this issue, the National Institute of Standards and Technology (NIST) has started a standardization process for quantum-resilient (a.k.a., post-quantum) cryptographic schemes that can survive quantum cryptanalysis [6]. This standardization is an ongoing process that is currently in its final phase. The remaining finalists include 3 digital signature schemes [7], [8], [9] and 4 public-key encryption and key establishment protocols [10], [11], [12], [13]. There are also 8 candidates that are still being considered as alternatives [14].

Lattice-based cryptosystems are the most popular ones among Round-3 candidates. This can be mainly attributed to the theoretical promise of lattices and the practicality of the resulting cryptosystems in terms of hardware and software implementation efficiency. Several prior works have investigated the performance and area-cost efficiency of lattice cryptography (and particularly NIST candidates) through software and hardware implementations [15]. At the same time, there is a significant effort put into quantifying the theoretical security using classical and quantum computers [16]. By contrast, the works that focus on the side-channel security assessment of lattice-based cryptography implementations schemes are limited. Although theoretical security and performance have been the main criteria for the NIST selection process, side-channel analysis is also taken into account [17].

The side-channel attacks and countermeasures are especially important for the Round-3 finalists given that they are close to real-world deployment. These attacks allow extracting secret values from the implementation characteristics such as execution time, power consumption, and electromagnetic radiation [18]. An attacker can perform the attacks with only a few side-channel measurements monitored from the physical device. An extreme case of these attacks is called *single-trace attacks* where the adversary can extract the secrets with a single execution's measurement. These attacks are more dangerous than the ones needing multiple traces because they can evade popular defenses such as masking [19].

Prior works on *single-trace* side-channel analaysis of lattice cryptosystems have targeted the Number Theoretic Transform (NTT) [20], [21], [22], discrete Gaussian sampling [23], [24], [25], rejection procedure [26], polynomial multiplication [27], [28], [29], [30], message encoding/decoding [31], [19], [32], [33], [34], and other related components such as the hash function [34], [19]. The ω-small polynomial sampling sub-routine is, however, overlooked. Several NIST Round-3 candidates use this operation during the key generation, key encapsulation, and signature generation processes.

In this paper, we propose single-trace side-channel attacks on the ω-small polynomial sampling of NTRU, NTRU Prime, and CRYSTALS-DILITHIUM reference software implementations. Our attack is the first one to target ω-small polynomial sampling and thus expose an orthogonal vulnerability compared to earlier works. We identify multiple types of side-channel vulnerabilities that exist in the reference implementations of NIST Round-3 candidates for the key establishment (NTRU and NTRU Prime) and digital signature scheme (CRYSTALS-DILITHIUM). We then present an attack that exploits the vulnerabilities in these implementations to recover

the secret and session keys.

Our attack targets the one-time polynomials used in key generation and encapsulation of NTRU/NTRU Prime, and the signing procedure in CRYSTALS-DILITHIUM. First, we identify that the ω-small polynomial sampling software computing these one-time values contain side-channel vulnerable steps that can potentially leak the '-1', '0', or '+1' assignments made to polynomial coefficients. Then, we propose a novel attack strategy that locates those operations and exploits vulnerabilities. We demonstrate that the recovered variables allow extracting the secret and session keys of NTRU and NTRU Prime, while reducing the entropy of the challenges for CRYSTALS-DILITHIUM's signatures. Finally, we quantify our attack success rate on the reference software implementations and illustrate its effect on the security of NTRU, NTRU Prime, and CRYSTALS-DILITHIUM cryptosystems.

The contributions of this paper are as follows.

- We propose the first side-channel attack on ω-small polynomial sampling software and show that the attack is applicable to two NIST Round-3 finalists and one alternative candidate.
- We reveal that the ω-small polynomial sampling implementations of NIST submissions contain side-channel vulnerabilities within their sub-routines (sorting in NTRU/NTRU Prime and shuffling in CRYSTALS-DILITHIUM) that can leak valuable information. We identify those regions of interests from the power measurements and show that the leakages can allow an adversary to recover the secret key and session key.
- We apply the proposed attack on the reference software of the three NIST Round-3 candidates taken from NIST's website and execute the attack on an off-the-shelf device containing an ARM-Cortex-M4 microcontroller. Our attacks extract the targeted coefficients with a success rate of 99.78% for NTRU and NTRU Prime, and 99.99% for CRYSTALS-DILITHIUM. This success rate reduces the brute-force search space of finding the secret key to 2^{41} or below for NTRU and NTRU Prime, and reduces rejected challenge polynomials' entropy between 39 to 60 bits for CRYSTALS-DILITHIUM.

Our results show that both sampling techniques leak information through single-trace side-channels but the vulnerability in CRYSTALS-DILITHIUM is less significant. The software developers including the NIST proposer teams, therefore, should consider such leakages and incorporate some defenses that can address the vulnerability.

II. Preliminaries

A. ω-small Polynomial Sampling

This work focuses on the generation of polynomials whose coefficients are $-1, 0$, or $+1$ and whose number of non-zero coefficients are defined as a part of the protocol parameters. Throughout the paper, we call those polynomials as ω-small polynomials where ω represents the non-zero coefficients' amount, which is smaller than $\frac{2}{3}$ of the degree of the polynomial ring. Several NIST post-quantum project candidates use ω-small polynomials. As part of their proposal to the standardization project, two algorithms have been defined to generate random ω-small polynomials.

Algorithm 1 Generation of random ω-small polynomials with small ω: SampleSparse()

Require: Number of non-zero coefficients (ω), degree of the ring (N).
Ensure: Random ω-small polynomial a.

```
initialize coefficients of c with 0
for i ← N − ω to N do
    repeat
        b ←$ Random(0, N)
    until b < N − ω
    a_i ← a_b
    a_b ←$ {−1, 1}
end for
return a
```

Algorithm 2 Generation of random ω-small polynomials: SamplePerm()

Require: predefined polynomial p with ω non-zero coefficients, degree of the ring (N).
Ensure: Random ω-small polynomial a.

```
a ← p
P ←$ {0, 1, ..., 2^30 − 1}^N
Permute a while sorting P
return a
```

TABLE I
Parameter sets of NTRU.

Scheme	security level	N	q	ω
ntruhps2048509	$-(1)$	509	2048	254
ntruhrss701	1(3)	701	8192	–
ntruhps2048677	1(3)	677	2048	254
ntruhps4096821	3(5)	821	4096	510

Algorithm 1, $SampleSparse()$, shows the first technique, which uses a modified, "inside-out", version of Fisher-Yates shuffling [35]. The algorithm first selects a random position between 0 and the degree of the input polynomial and swaps the coefficient at that position with a coefficient that is known to be zero. The algorithm then randomly assigns the same coefficient to either -1 or 1. This is repeated ω times so that the resulting polynomial has ω non-zero coefficients.

Algorithm 2, $SamplePerm()$, is the second technique to generate ω-small polynomials when ω is closer to $\frac{2N}{3}$. It generates a predefined ω-small polynomial and then permutes its coefficients with a sorting algorithm. The input predefined polynomial p is usually selected as its first ω coefficients to be 1 or -1, while the remaining coefficients are set to 0. The algorithm, then, generates an array with N elements that are uniformly random 30-bit integers and replicates swap operations for coefficients of a while sorting the random array.

B. NTRU and NTRU Prime

NTRU is a public-key encryption and key-establishment algorithm, which allows to safely transfer a session key between two (or more) parties over an insecure medium. NTRU

Algorithm 3 Key Generation

Require: $seed$
Ensure: $(pk, sk) = (h, (f, f_3, h_q, s))$.

1: $f \xleftarrow{\$} Ternary(seed)$,
2: $g \xleftarrow{\$} SamplePerm(seed)$
3: $f_q \leftarrow (1/f) \mod \mathbb{Z}/(X^{n-1} + X^{n-2} + \cdots + 1)$
4: $h \leftarrow (3 \cdot g \cdot f_q) \mod \mathbb{Z}_q/(X^n - 1)$
5: $h_q \leftarrow (1/h) \mod \mathbb{Z}_q/(X^{n-1} + X^{n-2} + \cdots + 1)$
6: $f_3 \leftarrow (1/f) \mod \mathbb{Z}_3/(X^{n-1} + X^{n-2} + \cdots + 1)$
7: $s \xleftarrow{\$} \{0,1\}^{256}$
8: **return** $(h, (f, f_3, h_q, s))$

Algorithm 4 Encapsulation

Require: $pk = h$
Ensure: $C = (c, k)$

1: $coins \xleftarrow{\$} \{0,1\}^{256}$
2: $r \xleftarrow{\$} Ternary(coins)$
3: $m \xleftarrow{\$} SamplePerm(coins)$
4: $c \leftarrow r \cdot h + Lift(m) \mod \mathbb{Z}_q/(X^n - 1)$
5: $k \leftarrow H_1(r, m)$
6: **return** (c, k)

is one of the four remaining finalists of NIST post-quantum project [36]. The submission consists of two versions, namely NTRU-HPS and NTRU-HRSS: while NTRU-HPS has three parameter sets, NTRU-HRSS has a single parameter set. In this work, we will focus on the NTRU-HPS that uses ω-small polynomials during the secret and session key generation. Proposed parameters for NTRU are given in Table I.

The NTRU scheme defines several parameters: for all parameter sets, n is a prime number where both 2 and 3 are order of $n-1$ in $(\mathbb{Z}/n)^\times$, the multiplicative group of integers modulo n, and q is power of 2. $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_r$, and $\mathcal{L}_m$ are sets of integer polynomials defined for f, g, r, and m respectively. $Lift$ is an injection $\mathcal{L}_m \to \mathbb{Z}[x]$ for NTRU-HRSS and the identity thus $Lift(m) = m$ for NTRU-HPS. In NTRU-HPS, $\mathcal{L}_f$ and $\mathcal{L}_r$ defined as non-zero polynomials, whose coefficients are in $-1, 0, 1$, of degree at most $n-2$ and $\mathcal{L}_g$ and $\mathcal{L}_m$ defined as a subset of $\mathcal{L}_f$, consisting of polynomials that have exactly $\omega/2$ coefficients equal to $+1$ and $\omega/2$ coefficients equal to -1. NTRU-HPS also defines $Ternary()$ function to generate random polynomials in $\mathcal{L}_f$ and uses $SamplePerm()$ function to generate random polynomials in $\mathcal{L}_g$.

Algorithm 3 shows the steps of the Key Generation procedure in NTRU-HPS. The algorithm takes in a random $seed$ and calculates the public key pk and the secret key sk needed for the later steps of key establishment. This algorithm first generates the secret key polynomials f and g using $Ternary()$ and $SamplePerm()$ functions, where $Ternary()$ randomly selects each coefficient from $\{0, 1, 2\}$ at uniform, and $SamplePerm()$ uses Algorithm 2. Then, it computes two inverses of f as f_q, f_3 and calculates the public key $h = (3 \cdot g \cdot f_q) \mod \mathbb{Z}_q/(X^n - 1)$ as well as its inverse h_q. The definition of $\mathcal{L}_g$ and $\mathcal{L}_m$ ensures that both $h \equiv 0 \pmod{(\mathbb{Z}_q/(X-1))}$ and $c \equiv 0 \pmod{(\mathbb{Z}_q/(X-1))}$. Therefore the polynomial inverses are computed modulo $(X^n - 1)/(X - 1) = X^{n-1} + X^{n-2} + \cdots + 1$.

TABLE II
PARAMETER SETS OF CRYSTALS-DILITHIUM.

security level	2	3	5
q	8380417	8380417	8380417
ω	39	49	60
(k, l)	$(4, 4)$	$(6, 5)$	$(8, 7)$
$log\binom{256}{\omega} + \omega$	192	225	257

Algorithm 4 shows the steps of encapsulation in NTRU-HPS. The algorithm takes in the public key pk and calculates the ciphertext c and the shared session key k. It first generates fresh r and m at random from $\mathcal{L}_r$ and $\mathcal{L}_m$ again using $SamplePerm()$ and $Ternary()$, respectively. Then, it determines the ciphertext $c = r \cdot h + m$ modulo $\mathbb{Z}_q/(X^n - 1)$. The shared key is computed as $k = H_1(r, m)$.

The decapsulation gets the ciphertext c and the secret key sk, and it returns a key k_2 that is equal to the encapsulated, shared session key k if the ciphertext is valid. We omit the details of decapsulation for brevity as it is not needed/targeted with our attack.

Although we describe NTRU, $SamplePerm()$ function is also used in the streamlined NTRU Prime, which is part of another NIST candidate, NTRU Prime. This protocol likewise calls $SamplePerm()$ function for creating f and r polynomials [36]. Since both algorithms sample and use the polynomials with the same manner, we show just one of them in this paper for brevity.

C. CRYSTALS-DILITHIUM

Digital signature schemes generate a valid signature on a message using a secret key and the signature's authenticity can be verified with the associated public key. CRYSTALS-DILITHIUM is one of the three finalists running for the NIST's post-quantum digital signature standard. The scheme uses matrices and vectors of polynomials in $R = \mathbb{Z}_q/(X^{256} + 1)$. CRYSTALS-DILITHIUM defines 10 parameters but 3 of them are related to input/output of the $SampleSparse()$ function. Thus we list only those parameters in Table II for brevity. The last line of the table provides the theoretical challenge entropy for the output of $SampleSparse()$ function for ω-small polynomials.

Algorithm 5 shows the Key Generation operations that computes the secret key sk and the public key pk. This algorithm first generates a bitstring θ uniformly random and uses it to derive ρ, ζ, and K where ρ used to derive the public polynomial A, K is the part of sk and ζ is used to derive the secret key polynomials s_1 and s_2. Then, it computes $t = As_1 + s_2$. $Power2Round$ function splits coefficient-wise most and least significant bits of t as t_1 and t_0, respectively. While t_1 becomes part of the public key, t_0 is treated as a component of the secret key. H is instantiated as $SHAKE - 256$ hash function and CRH is a collusion resistant hash function.

Algorithm 5 Key Generation

Ensure: $(pk, sk) = ((\rho t_1), (\rho, K, tr, s_1, s_2, t_0))$.

1: $\theta \leftarrow \{0,1\}^{256}$,
2: $(\rho, \zeta, K) \in \{0,1\}^{3\times 256} \leftarrow H(\theta)$
3: $(s_1, s_2) \in S_\nu^l \times S_\nu^k \leftarrow H(\zeta)$
4: $A \in R_q^{k\times l} \leftarrow ExpandA(\rho)$
5: $t \leftarrow As_1 + s_2$
6: $(t_1, t_0) \leftarrow Power2Round(t, d)$
7: $tr \in \{0,1\}^{384} \leftarrow CRH(\rho||t_1)$
8: **return** $(pk = (\rho, t_1), sk = (\rho, K, tr, s_1, s_2, t_0))$

Algorithm 6 Sign

Require: sk, M
Ensure: $\sigma = (z, h, \tilde{c})$

1: $A \in R_q^{k\times l} \leftarrow ExpandA(\rho)$
2: $\mu \in \{0,1\}^{384} \leftarrow CRH(tr||M)$
3: $\kappa \leftarrow 0, (z, h) \leftarrow \perp$
4: $\rho' \in \{0,1\}^{384} CRH(K||\mu)$
5: **while** $(z, h) \perp$ **do**
6: $\quad y \in S_{\gamma_1}^l \leftarrow ExpandMask(\rho, \kappa)$
7: $\quad \tilde{c} \in \{0,1\}^{256} \leftarrow H(\mu, HighBits_q(Ay, 2\gamma_2))$
8: $\quad c \in B_\omega \leftarrow SampleSparse(\tilde{c})$
9: $\quad z \leftarrow y + cs_1$
10: $\quad$ **if** $||z||_{\gamma_1-\beta}$ or $||LowBits_q(Ay - cs_2, 2\gamma_2)||_{\gamma_2-\beta}$ **then**
11: $\quad\quad (z, h) \leftarrow \perp$
12: $\quad$ **else**
13: $\quad\quad h \leftarrow MakeHint_q(-ct_0, Ay - cs_2 + ct_0, 2\gamma_2)$
14: $\quad\quad$ **if** $||ct_0||_{\gamma 2}$ of 1's in h is greater than h_w **then**
15: $\quad\quad\quad (z, h) \leftarrow \perp$
16: $\quad\quad\quad \kappa \leftarrow \kappa + l$
17: **return** $\sigma \leftarrow (z, h, \tilde{c})$

Algorithm 6 gives the steps of Sign algorithm, which creates the signature σ on an input message M by using the secret key sk. The algorithm uses rejection sampling: it generates uniformly random y and ω-small polynomial c, checks if they reveal any information about secret key, and calculates new y and c till it finds proper candidates that does not reveal any information about the secret key. When the algorithm finally finds such c and y values, it creates the signature $z = y + cs_1$, $h = MakeHint_q(-ct_0, Ay - cs_2 + ct_0, 2\gamma_2)$, and $\tilde{c} = H(\mu, HighBits_q(Ay, 2\gamma_2))$, where $MakeHint_q$ determines the distance caused by t_0 and $HighBits_q$ returns high-order bits of each coefficients of the input.

The generated signature can be verified using the public key pk. We omit the details of this verification process for brevity since it is not needed/targeted with our attack.

D. Threat Model

Our attack follows the standard assumptions in single-trace side-channel attacks [37], [20]. First, the attacker has physical access to the identical device and can configure it with known-keys to capture power traces. Therefore, the adversary can capture multiple measurements to create templates corresponding to known keys or intermediate values. Second, we assume that the adversary has sufficient knowledge about the details of the target software implementation, i.e., the adversary downloads and inspects the publicly available software packages of candidates submitted to NIST. When running the attack, the adversary is limited to a single measurement. Note that this has two advantages for the attacker: (1) key generation and encapsulation can be targeted because they use one-time values and (2) the masking defenses become ineffective [19]. In our scenario, the attacker targets "one-time" variables used in NTRU-HPS, NTRU Prime, and CRYSTALS-DILITHIUM. These occur during NTRU Prime's and NTRU-HPS's key generation for extracting secret keys, and also encapsulation for extracting session keys, and during CRYSTALS-DILITHIUM's signing.

III. The Proposed Attacks

This section presents the proposed attack strategy for recovering the secret and session keys on NTRU-HPS (and Streamlined NTRU Prime) and CRYSTALS-DILITHIUM schemes. We will first describe the intermediate computations we target and why extracting those variables enables recovering session or secret keys. We then analyze how those variables are generated and identify vulnerabilities.

A. The Attack on NTRU Prime and NTRU-HPS

Our attack targets the random polynomial sampling within the NTRU-HPS key generation shown in Algorithm 3 line 5 and encapsulation in Algorithm 4 line 3. We argue that (i) a single-trace side-channel attack on these generated polynomials g and m is possible and (ii) that capturing the coefficients of those polynomials with the side-channel attack allows the adversary to recover the secret key and session key. Since both NTRU-HPS and Streamlined NTRU Prime employ the same function to sample polynomials, we demonstrate our attack steps only on NTRU-HPS for simplicity. The proposed attack is applicable for both algorithms.

1) The Targeted Operation SamplePerm() and Rationale

We argue that an adversary can recover the secret key in NTRU-HPS by targeting the random polynomial generation during NTRU-HPS key generation phase. NTRU-HPS key generation algorithm returns a secret key sk and a public key pk. The secret key consist of two random polynomials. One of them is in $\mathcal{L}_f$ and the other one is in $\mathcal{L}_g$. NTRU-HPS key generation algorithm uses these two random polynomials to generate public key pk relying on the equation $h = 3g/f \in \mathcal{R}/q$ where h is the public key pk (see Algorithm 3 line 6). The secret key sk hardness also relies upon the secrecy of the polynomial f. Therefore, if the adversary knows the polynomial g, it can reverse public key equations $h = g/(3f)$, derive the polynomial f, and successfully rebuild the secret key sk.

Another key observation is that the random polynomial generation exists in NTRU-HPS encapsulation implementation as well. Therefore, the attacker can extract the generated random polynomial m and then reconstruct the session key

Listing 1. NTRU Sorting Reference Implementation

```
1  void crypto_sort_int32(
2          int32 *array, size_t n)
3  {
4    ...
5    for (p = top;p >= 1;p >>= 1){
6      i = 0;
7      while (i + 2 * p <= n){
8        for (j = i;j < i + p;++j){
9          int32_MINMAX(x[j],x[j+p]);
10       }
11       i += 2 * p;
12     }
13   ...
14 }
```

Listing 2. NTRU Comparison Reference Implementation

```
1  #define int32_MINMAX(a,b) \
2  do { \
3    int32_t ab = (b) ^ (a); \
4    int32_t c = \
5        (int32_t)((int64_t)(b) \
6        - (int64_t)(a)); \
7    c ^= ab & (c ^ (b)); \
8    c >>= 31; \
9    c &= ab; \
10   (a) ^= c; \
11   (b) ^= c; \
12 } while(0)
```

using the known h and c. The encapsulation algorithm first generates a random polynomial m with the $SamplePerm()$ function (see Algorithm 4). This polynomial m is also known as the message that the encapsulation algorithm uses to create the session key. The polynomials m and r are the private elements needed to generate the session key of encapsulation algorithm. This can be achieved by hashing $r = (c - m) \cdot h_q \mod \mathbb{Z}_q/(X^{n-1} + X^{n-2} + \cdots + 1)$ and m where h_q is the inverse of the public key (see line 5 of Algorithm 4).

We argue that targeting this random polynomial generation requires performing a *single-trace* side-channel attack. Indeed, these algorithms generate fresh random polynomials so the attacker can acquire only a *single* power measurement trace for the targeted polynomials g and m. We next conduct analysis for the single-trace side-channel vulnerability of SamplePerm that generates these polynomials.

2) Identifying the Vulnerabilities of `crypto_sort`

The random polynomial generation follows two subroutines: sampling and sorting as shown in Algorithm 2. The sorting algorithm follows a permuting sequence to determine the secret information. The permuting sequence strategy is as follows. First, the polynomial generation algorithm gets a predefined polynomial p as input, which is the polynomial with the first $\omega/2$ coefficients are 1, the next $\omega/2$ coefficients are 2, and other coefficients are 0. Second, the sequence generates a uniformly random polynomial P whose coefficients are 30-bits—these coefficients gets multiplied by 4. Third, the coefficients of secret polynomial, which are in $\{0, 1, 2\}$, encoded into the least significant 2-bits of polynomial P. Fourth, the polynomial P is sorted as if it consists of 32-bit integers. Fifth, coefficients of secret polynomials extracted

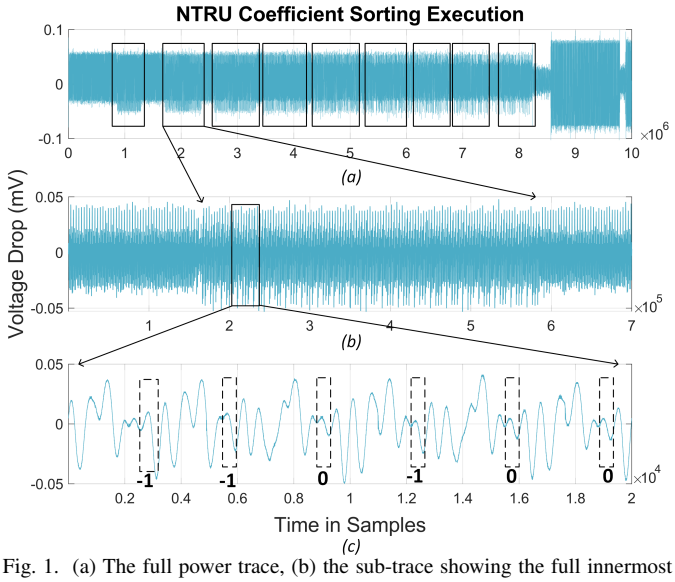

Fig. 1. (a) The full power trace, (b) the sub-trace showing the full innermost loop, (c) the sub-trace for the 6 iterations of the innermost loop. The regions of interests are detectable from the full power measurement and the comparison results can be identified.

from P's least significant 2-bits. Since the poly p is predefined and public in the implementation, recovering the secret a requires attacking the comparison operation and figuring out the sorting order. Therefore, our attack targets the permuting sequence's implementation with the goal of extracting the generated polynomial a.

Listing 1 shows the NTRU-HPS scheme's reference implementation of sorting algorithm in C, which is obtained from the Round-3 NIST submission package. Note that although the `crypto_sort_int32` function has other steps in its implementation, these steps are not shown for brevity as they are not needed to describe our attack. The `crypto_sort_int32` function requires two inputs: a polynomial and its size. Then, this function compares the two coefficients at a time and replaces the smaller one's location with the big one. This is repeated for all the coefficients of the polynomial such that upon termination of this loop the entire polynomial is sorted. The `int32_MINMAX` function performs this comparison operation.

The Listing 2 shows the NTRU HPS's `int32_MINMAX` function implementation in reference software. This function sorts the two inputs such that the first element is less than or equal to the second element. If the two inputs are already in this order, the function should not perform the swapping. Else, if the first input is greater, the contents are swapped. The function performs this sorting in a sequence of logical operators and in constant time in several steps. The two input coefficients first go through logical operations such as XOR, AND, and OR. Then, a logical shift operation at line 8 decides if the first coefficient is greater than the second coefficient. If this is the case, the shift operation then results in -1. Otherwise, it returns 0. If the attacker is able to separate these two cases, it can recover the coefficients since the input polynomial is known.

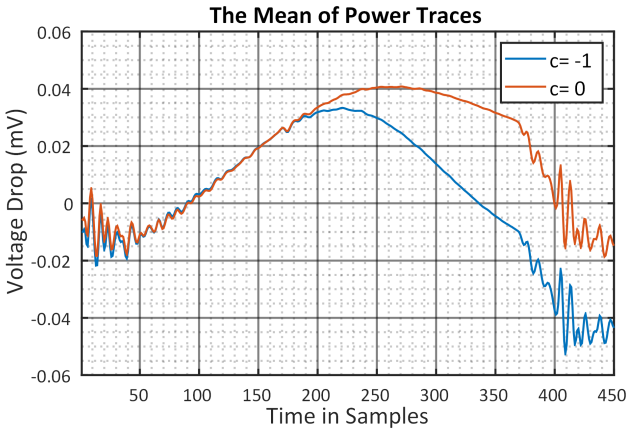

Fig. 2. The mean of sub-traces for different comparison results. There is a significant power consumption difference between these comparison results.

3) Pinpointing Regions of Interest

A major challenge in performing our proposed attack is identifying and isolating each coefficient's individual execution. NTRU-HPS encapsulation algorithm calls the comparison function multiple times within nested loops as shown in Listing 1. The attacker, therefore, has to locate the innermost loop to attack the `int32_MINMAX` function. Figure 1 (a) plots the power trace for `crypto_sort_int32`. The rectangles show multiple iterations of the second while-loop (Listing 1 line 7), while Figure 1 (b) shows the execution of for-loop shown at the line 8 in Listing 1. Figure 1 (c) further depicts the power consumption of six iterations of this for-loop. It means that there are six `int32_MINMAX` function calls. The black dashed lines correspond to the targeted comparison operation (see line 7 in Listing 2). This figure illustrates that we can distinguish both sorting and comparison functions' power behaviors from other execution steps and that there is some observable variation based on the targeted shift operation result.

4) Inspecting the Vulnerability

We expect a significant power consumption difference between the two possible outputs 0 and -1 during the execution of the targeted shift operation at line 7 in Listing 2. This vulnerability occurs due to the significant difference in the Hamming weight (HW) representations. The -1 (0xFFFFFFFF) shows a power consumption behavior for HW 32, while the output 0 (0x00000000) behavior matches with HW 0. We next inspect if this predicted vulnerability can be observed on the power measurements. Figure 1 (c) demonstrates the difference with the dashed-lined rectangles. We then captured 100K traces, half is when the result is -1 and other half is for 0. We averaged the two power trace sets shown in Figure 2. The blue line represents the mean power consumption for result -1 and the orange line shows the averaged power consumption for result 0. We adopted an earlier technique [29] to isolate these cycles and synchronize all sub-traces based on their maximum and/or minimum power draw point. We did not use any other post-processing on the obtained sub-traces.

Figure 2 displays that the different comparison results have different power consumption character. This confirms our base hypothesis that a single-trace side-channel attack on the random polynomial generation is possible and feasible. Section IV provides more details on the success rate using statistics.

Listing 3. Dilithium Polynomial Generation Reference Implementation

```
1  void poly_challenge(poly *c,
2                      const uint8_t seed[SEEDBYTES])
3  {
4    ...
5    for(i = 0; i < 8; ++i)
6      signs |= (uint64_t)buf[i] << 8*i;
7    pos = 8;
8    for(i = 0; i < N; ++i)
9      c->coeffs[i] = 0;
10   for(i = N-TAU; i < N; ++i) {
11     do {
12       if(pos >= SHAKE256_RATE) {
13         shake256_squeezeblocks(buf, 1,
14                                &state);
15         pos = 0;
16       }
17       b = buf[pos++];
18     } while(b > i);
19     c->coeffs[i] = c->coeffs[b];
20     c->coeffs[b] = 1 - 2*(signs & 1);
21     signs >>= 1;
22   }
23 }
```

B. The Attack on CRYSTALS-DILITHIUM

Our attack targets random challenge's sampling during the signature generation given in Algorithm 6 line 8. We argue that (i) a single-trace side-channel attack on this challenge polynomial c is possible and (ii) that capturing the coefficients of those polynomials with the side-channel attack allows the adversary to recover the signing key.

1) The Targeted Operation SampleSparse() and Rationale

CRYSTALS-DILITHIUM scheme protects the secret key by rejecting the random challenge polynomials that expose the secret information. Hence, any information on the rejected challenge polynomials can also lead to information about the secret key. CRYSTALS-DILITHIUM creates the challenge polynomials with SampleSparse function in Algorithm 6 line 8.

2) Identifying the Vulnerabilities of SampleSparse()

The SampleSparse function in CRYSTALS-DILITHIUM follows Algorithm 1. This is implemented with the code shown in Listing 3. This implementation creates a polynomial whose only TAU number of coefficients are non-zero. The code first initializes the random polynomial's coefficients with zeros (see line 9 in Listing 3). Then, it replaces the TAU number of zero coefficients with either 1 or -1. This replacement is performed non-deterministically based on $shake256$ output. At line 17, the code decides which coefficient gets a non-zero value. Then, line 19 shows that the chosen coefficient value is preserved by assigning its value to another coefficient. Finally, the code updates the chosen coefficient value with a non-zero value (1 or -1) at line 20.

An adversary can target line 20 where the implementation determines the sign of the non-zero coefficients. This operation can leak information about how many negative and posi-

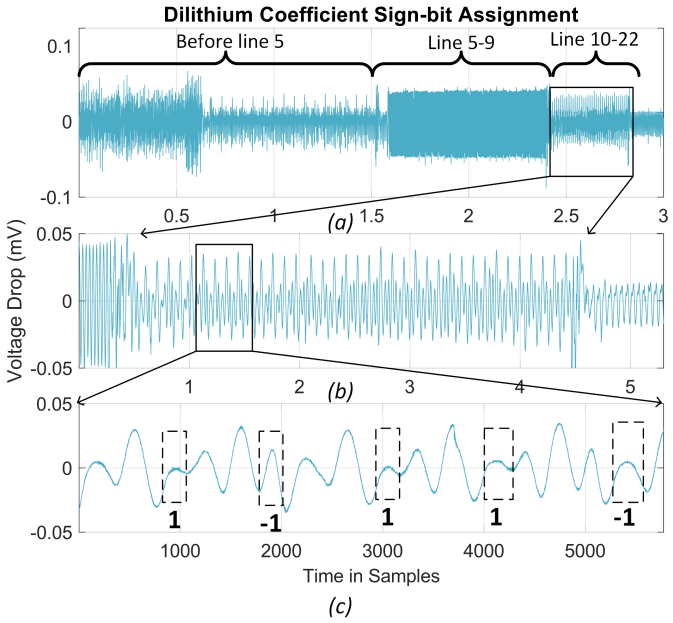

Fig. 3. (a) The full power trace, (b) the sub-trace showing the full for-loop, (c) the sub-trace for the 5 iterations of this for-loop. The regions of interests are detectable from the full power measurement and the coefficient assignments can be identified.

tive coefficients the private polynomial has, which gives a hint about the secret challenge. We illustrate how leveraging this information reduces the search space in Section IV.

Although line 17 may seem like a promising attack point, it actually is not the best option. This is because the targeted values (b) range between 0 and 255 and thus there are many possible candidates, which complicates the attack. Another misleading attack point is targeting line 19 where the implementation assigns 0, 1 or -1 to the coefficients whose indexes range between $255 - TAU$ and 255. The implementation here replaces the TAU number of zero coefficients non-deterministically. Hence, it might assign a fresh coefficient or re-use an assigned value and propagate this assigned value to a new coefficient. This is hard to track for the propagated coefficients and makes the attack complicated.

3) Pinpointing Regions of Interest

Identifying and isolating each coefficient's individual execution is again a challenge for this case just like as in NTRU-HPS. Although CRYSTALS-DILITHIUM performs the coefficient assignments in one loop instead of nested loops, the attacker still has to locate the coefficient assignment operations to perform the attack.

Figure 3 (a) shows the power trace for the implementation of SampleSparse algorithm in CRYSTALS-DILITHIUM. The figure shows three sub-traces divided by three braces. The first brace represents the last two out of five `shake256` function calls. The polynomial generation requires `shake256` functions to fill a buffer (`buf[256]`) with random numbers right before the line 5 in Listing 3. Listing 3 does not show the `shake256` functions for brevity since they are out of our attack's scope. The second sub-trace shows the power measurement corresponding to operations between lines 5

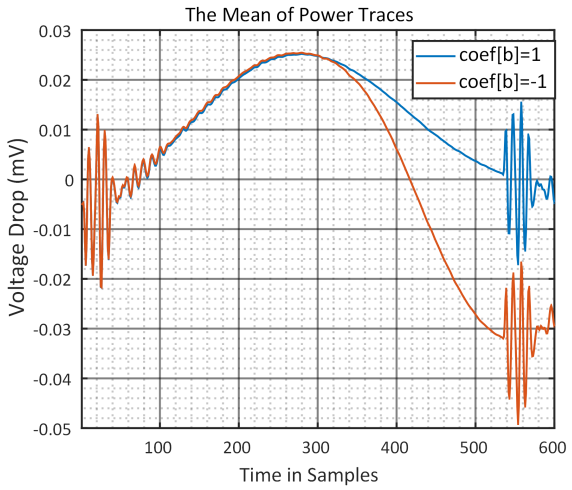

Fig. 4. The mean of sub-traces for different coefficient assignments. There is a significant power consumption difference between these assignments.

and 9, where the software creates a 64-bit random value and initializes all coefficients with zero. The last sub-trace, enveloped with a rectangle, exhibits the targeted for-loop execution (lines 10 and 22).

Figure 3 (b) shows the execution of for-loop shown with lines 10 and 22 in Listing 3, while Figure 3 (c) further depicts the power consumption of five iterations of this for-loop. It means that there are five non-zero coefficient assignments (Listing 3 line 20). The black dashed lines correspond to the targeted non-zero coefficient assignments. This figure illustrates that we can visually distinguish coefficient assignments' power behaviors from other execution steps and that there is 'some' observable variation based on -1 vs. 1 assignments.

4) Inspecting the Vulnerability

We target a coefficient assignment operation. The assignment possible outcomes are -1 (0xFFFFFFFF) and 1 (0x00000001); hence, we expect two significantly different power measurements due to the high HW difference. The expected vulnerability requires analyzing the power measurements related to the targeted operation. We, therefore, captured 100K traces for CRYSTALS-DILITHIUM's SampleSparse algorithm's implementation.. We then chopped traces into the parts to obtain sub-traces that exactly correspond to the power consumption during the execution line 20 in Listing 3. Since there are two possible outputs (1, -1) for line 20, there are two power trace sets. Figure 4 shows these averaged trace sets. The blue trace corresponds to the mean power consumption for 1, while the orange one is for -1.

Figure 4 displays that the different coefficient assignments have different power consumption character. This confirms our base hypothesis that a single-trace side-channel attack on the random polynomial generation is possible and feasible. Our proposed single-trace side-channel attack can extract the information about how many negative and positive coefficients exists in the challenge polynomial. Section IV provides more details on the success rate using statistics.

IV. Experiment Results

In this section, we first describe our experiment setup. We then present our single-trace template-based side-channel attack steps. Later, we evaluate leakage information obtained via the single-trace power side-channel attack. This evaluation consists of the success rate of the proposed attack. Finally, we analyze the effects of the proposed attack on the security of the schemes.

A. Attack Environment and Equipment

The device under attack is a development board that uses an ARM Cortex-M4F core operating at 30 MHz. We used the Round-3 submission package of the NIST reference software implantation for NTRU and CRYSTALS-DILITHIUM schemes. We compiled the code with `gcc-arm-none-eabi-4_8-2014q1` compiler tool and with `-O3` flag. We used PicoScope 3206D Oscilloscope and set the sampling rate at 1GS/s. The oscilloscope obtained measurements through the Tektronix CT1 passive current probe, whose bandwidth is 1–1000 MHz at 3 dB. We did not use any external amplification to improve the measurements.

B. Template Based Single-Trace Side-Channel Attack

Single-trace side-channel attacks aim to extract a secret value from one side-channel measurement. Since NTRU and CRYSTALS-DILITHIUM schemes execute the targeted random polynomial generation step once, we perform a single-trace template attack.

The summary of our attack strategy is as follows. We first analyze the entire trace to determine the sub-trace selection and alignment strategy. We then use sum-of-squared-differences (SOSD) method [38] to identify points-of-interests (POI) that leaks information. Next, we build uni-variate Gaussian template [39] with the selected POI and conduct a template attack. We finally use a large number of samples to build the templates and to test the accuracy of the attack, and we report the success rate of the attacks to quantify the vulnerability of reference implementation. The rest of this section walks the reader through these steps with illustrators.

$$M_{k,i} = \frac{1}{T_k} \sum_{j=1}^{T_k} t_{j,i} \quad , \tag{1}$$

$$SOSD_i = \sum_{k_1,k_2} (M_{k_1,i} - M_{k_2,i})^2 \quad , \tag{2}$$

The POI identification uses the SOSD technique described by Equations 1 and 2 where there are k different operations and i sample points in a number of traces $t_{1,i}...t_{k,i}$ used to calculate the mean power trace $M_{k,i}$ and the SOSD trace SOSD$_i$. One point stands out from the experiment and we performed an initial study to understand if there is any meaningful vulnerability.

$$\mu_i = \frac{1}{T_k} \sum_{j=1}^{T_k} t_{j,s_i} \quad , \tag{3}$$

$$v_i = \frac{1}{T_k} \sum_{j=1}^{T_k} (t_{j,s_i} - \mu_i)^2 \quad , \tag{4}$$

To quantify the empirical effect of a potential vulnerability, we ran a template attack using 90K measurements for profiling and 10K measurements for tests. Note that the attack runs independently on each of these 10K test measurements. We picked one peak point as POI where the variation of power consumption is maximum at this sample s_i. We choose a single POIs in our case but multiple POIs might be chosen to increase attack success rate for noisier platforms. For this POI, we calculated average power μ_i with Equation 3, variance of power v_i with Equation 4.

$$P_k = \sum_{j=0}^{k} \log \mathcal{N}(t_{j,s_i}, \mu_i, v_i) \quad , \tag{5}$$

We build the template from normal probability density function (NPDF) using the POI of the attacked traces t_{j,s_i}, μ_i and v_i results of the profiling. We summed the log of the normal distribution $\mathcal{N}$ with Equation 5 to avoid precision issues that occur if the results of NPDF are too large or too small. The index of the matrix P_k with the highest value corresponds to the predicted coefficient.

C. Quantifying the Success Rate

We applied the single-trace side-channel attack defined in Section III for NTRU-HPS and CRYSTALS-DILITHIUM schemes. Although both algorithms require random polynomials, they generate their polynomials in different ways. Thus, we have different profiles and test sets for the two implementations.

For NTRU-HPS encryption scheme, we targeted the comparison operation result at line 8 in Listing 2. This operation ends with either 0 or -1. Hence, we have two groups both for profiling and testing. Figure 2 displays two averaged power traces for these two comparison outputs. Indeed, there is a significant power consumption difference between the sample points 220 and 400. Using the SOSD method, we chose the sample point 375 as our POI. Afterward, we captured 100K traces, 90K measurements for profiling and 10K measurements for tests. Evaluating on this 10K tests reveals that our attack can achieve 99,98% success rate for recovering the secret polynomial.

Although our test result empirically gives 99,98% success rate, we quantify the attacker success rate by modeling the power distribution to derive a theoretical estimate. Figure 5 illustrates the distribution of 100K power values of both 0 and -1 responses at the sample point 375 (i.e., POI). The blue line corresponds to the power distribution of comparison result -1 with standard deviation (σ) 0.004 and mean (μ) -0.054, while the orange distribution curve represents comparison result 0 and its σ, μ pair is 0.005, -0.023.

Figure 5 also shows that the power distributions intersect approximately between -0.42 mV and -0.038 mV. If the measured power trace voltage is between in this range at the

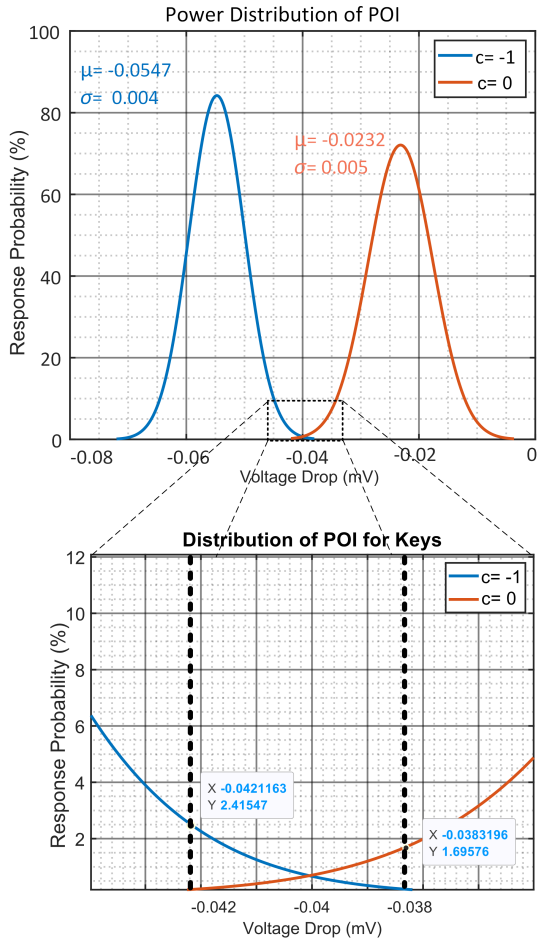

Fig. 5. The distribution of power measurements at the POI for the two different assignments (-1 vs. 0) during the targeted instruction in NTRU-HPS. These two power distributions intersect with each other and the intersection reduces the success rate.

given POI, the proposed attack might mispredict the result. The attack success rate, therefore, depends on the intersection area that remains between the black dash lines and under the distribution curves. We use Equation 6 to calculate the intersection area shown in Figure 5. The function $D(x)$ gives the integral of the normal distribution from the $-\infty$ to x where the $-\infty$ and x represent the interval values.

$$D(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-(x'-\mu)^2/(2\sigma^2)} dx' \quad , \tag{6}$$

We use this equation to find the intersection area by choosing the intervals as distributions intersect. The first interval value range is $[-0.42, \infty]$ since the first intersection's x-coordinate is 0 mV, while the second calculation uses $[-\infty, -0.038]$ as the interval range. After separately calculating the two regions under each distribution curve and adding them together, we obtain the theoretical failure rate, which corresponds to 99.78%. Note that this theoretical estimate is more pessimistic than our actual, empirical success rate.

For CRYSTALS-DILITHIUM signature scheme, we targeted the coefficient assignment step line 20 in Listing 3. The coefficient assignment has two possible outputs 1 or -1. Figure 3 shows that there is a significant power consumption

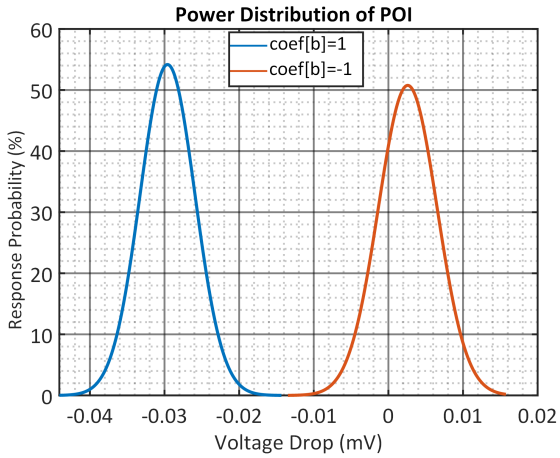

Fig. 6. The distribution of power measurements at the POI for the two different assignments (-1 vs. 1) during the targeted instruction in CRYSTALS-DILITHIUM. Although the two power distributions theoretically intersect with each other, the intersection cannot be seen visually.

difference between the sample points 350 and 530 during the execution of coefficient assignment[1]. Using the SOSD method, we chose the sample point 514 as our POI. For this attack point, our test result shows that the attack success rate is 100%. We can perfectly classify the -1 and 1 coefficients through power measurements, while the theoretical success rate is 99.9974% based on the power distribution shown in Figure 6.

D. The Effect of Success Rate on the Entropy

Table III shows claimed security of all NTRU-HPS scheme in non-local and local (in parenthesis) assumptions as defined in [10], number of calls to Listing 2, and the expected entropy for a full secret/session key recovery. Our attack achieves a 99.78% success rate per each call and the number of calls ranges from 9665 to 18493 depending on the parameters. Since we can identify each of those potentially incorrect classification using the intersection in Figure 5, we can recover correct values with a brute-force search following the side-channel attack. Therefore, the remaining entropy (i.e., security level) reduces to 2^{22}–2^{41} after our attack. Figure 7 depicts the success rate vs. the remaining entropy for the ntruhps4096821.

TABLE III
ENTROPY OF A FULL KEY EXTRACTION ATTACK TO NTRU-HPS.

Scheme	Claimed security	# calls Listing 2	Entropy
ntruhps2048509	105(139)	9665	22
ntruhps2048677	144(205)	14473	32
ntruhps4096821	178(253)	18493	41

On CRYSTALS-DILITHIUM, our attack recovers the assignments made to the coefficients (-1 or +1). Therefore, it removes the effect of τ from the logarithms (last line of the Table II). After our attack, the entropy of c would be reduced from 192, 225, and 257, to 153, 176, and 197 for NIST levels 2, 3, and 5, respectively. Although this can cause a concrete

[1]The captured trace number is 100K for CRYSTALS-DILITHIUM signature scheme as well with the same 90K profiling and 10K test split.

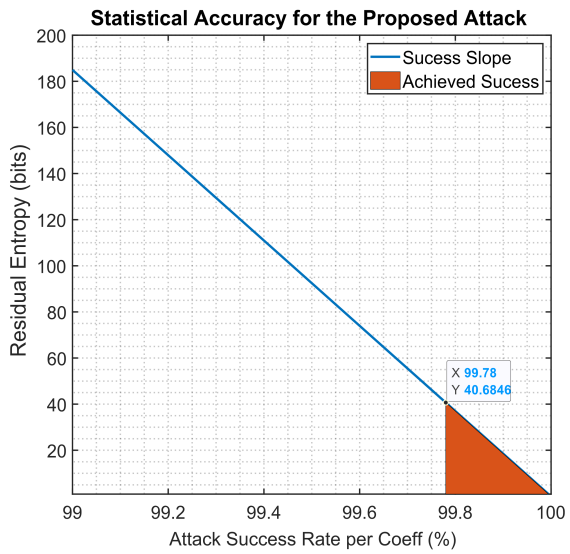

Fig. 7. The success rate of our proposed attack vs. the residual entropy of the session/secret keys in NTRU-HPS. Our per-coefficient success rate of 99.78% reduces the search space of keys to about 2^{41}.

reduction in the security, it is not as direct or pronounced as the attack on NTRU/NTRU Prime. Hence, our work quantifies that the sampling approach taken in CRYSTALS-DILITHIUM is more single-trace side-channel resilient than the one in NTRU/NTRU Prime.

V. Discussions

In this section, we discuss the related issues and comment on the drawback of our attack.

A. Applicability to Other Implementations

Our proposed attack is also applicable to the assembly optimized NTRU-HPS, NTRU Prime, and CRYSTALS-DILITHIUM or their implementations in the pqm4 library. NTRU-HPS and NTRU Prime use SUPERCOP [40] crypto sort C implementation for $SamplePerm()$ function, while pqm4 uses a different function but following essentially the same approach. Although we used the reference C implementation of the Round-3 NIST submission package as our target implementation, the targeted vulnerability thus also exists in C and assembly implementations of the pqm4 library. Likewise, the software implementation of the $SampleSparse()$ function in CRYSTALS-DILITHIUM NIST submission package is reused in pqm4 library. Our proposed attack, therefore, is suitable both for CRYSTALS-DILITHIUM NIST submission and pqm4 library implementations.

B. Calibration Factors of the Experiments

The noise of the platform decreases with the operating frequency of the device. We set the lowest design frequency of the development board, i.e., 30 MHz, to reduce the noise. Note that there are multiple prior works on single trace side-channel attacks demonstrated with even lower frequency such as 7.38 MHz to attack polynomial multiplication in NTRU-HPS scheme [28] or 8 MHz to attack NTT [21]. Our clock frequency is higher than these works. If an even higher frequency is analyzed, a better equipment for obtaining power measurements, an additional probe for detecting near-field electromagnetic leakages, or amplification/post-processing for noise reduction may be needed.

C. Drawbacks of Our Attack

Template attack has some well-known limitations and challenges, such as the cross-device and processing time limitations. In our scenario, we chose one POI for each attack point which led to a reasonable success rate and take minutes to build the profiles. More POIs can further improve the attack's success rate while needing more processing time. We limit our attack to a single device, cross-device attacks may need a more complicated, machine-learning based profiling [41], [42]. We note that this cross-device limitation of single-trace side-channels of post-quantum cryptosystems has been an open problem existing in several prior works [23], [21], [33], [25], [30], [27].

VI. Conclusions and Future Work

Although lattice cryptography is a popular and versatile tool providing quantum-resilience at a reasonable cost, it contains unique operations that have not been rigorously analyzed for side-channel vulnerabilities. This paper demonstrated the first single-trace side-channel attack on such an operation—ω-small polynomial sampling—used in lattice cryptography. We focused on two different algorithms' implementations that samples ω-small polynomials. We revealed that the specific sub-routines employed in NIST finalists/alternatives including NTRU, NTRU Prime, and CRYSTALS-DILITHIUM are indeed vulnerable to a power-based side-channel attack. Specifically, we showed that $SamplePerm$ implementation in NTRU can reveal positions of the coefficients with respect to their default position, while $SampleSparse$ implementation in CRYSTALS-DILITHIUM could only reveal information about the signs of non-zero coefficients, making it more secure against single-trace side-channel attacks. The results confirm the practicality of our attack on an off-the-shelf device.

The vulnerability we expose is orthogonal to the prior single-trace attacks and, by definition, to multi-trace attacks. Therefore, the defenses proposed/patched for other vulnerabilities have to be re-evaluated as they will likely fail against this specific leakage.

We emphasize that this an attack paper and that our goal is to inform the developers of these algorithm implementation about the vulnerabilities they introduce. Some form of countermeasure is, therefore, needed to address these attacks, e.g., based on shuffling or other forms of randomization or via differential circuit styles. More broadly, we argue that our paper highlights the need for single-trace side-channel aware software development.

VII. Acknowledgements

This research is supported in part by the National Science Foundation under Grant No. 1850373. NCSU is an academic partner of Riscure Inc., and we thank them for providing hardware/software support for side-channel analysis.

REFERENCES

[1] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *arXiv preprint quant-ph/0301141*, 2003.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[3] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016, vol. 12.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[6] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016, vol. 12.

[7] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[8] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, 2018.

[9] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 164–175.

[10] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, T. Saito, J. M. Schanck, P. Schwabe, W. Whyte, K. Xagawa, T. Yamakawa, and Z. Zhang, "NTRU: round 3," Submission to the NIST PQC standardization process, URL: https://ntru.org/index.shtml, 2019.

[11] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem," in *International Conference on Cryptology in Africa*. Springer, 2018, pp. 282–305.

[12] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE EuroS&P*. IEEE, 2018, pp. 353–367.

[13] D. J. Bernstein, T. Chou, T. Lange, R. Misoczki, R. Niederhagen, E. Persichetti, P. Schwabe, J. Szefer, and W. Wang, "Classic mceliece: conservative code-based cryptography 30 march 2019," 2019.

[14] NIST, "Round 3 submissions - post-quantum cryptography: Csrc," Submissions to the NIST PQC standardization process, URL: ://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions, 2021. [Online].

[15] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–41, 2019.

[16] J. Krämer and P. Struck, "Encryption schemes using random oracles: from classical to post-quantum security," in *International Conference on Post-Quantum Cryptography*. Springer, 2020, pp. 539–558.

[17] D. Moody, "The 2nd round of the NIST PQC standardization process," https://csrc.nist.gov/CSRC/media/Presentations/the-2nd-round-of-the-nist-pqc-standardization-proc/images-media/moody-opening-remarks.pdf.

[18] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[19] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked ind-cca secure saber kem."

[20] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 513–533.

[21] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2019, pp. 130–149.

[22] I.-J. Kim, T.-H. Lee, J. Han, B.-Y. Sim, and D.-G. Han, "Novel single-trace ml profiling attacks on nist 3 round candidate dilithium," 2020.

[23] S. Kim and S. Hong, "Single trace analysis on constant time cdt sampler and its countermeasure," *Applied Sciences*, vol. 8, no. 10, p. 1809, 2018.

[24] T. Espitau, P.-A. Fouque, B. Gérard, and M. Tibouchi, "Side-channel attacks on bliss lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1857–1874.

[25] C. Zhang, Z. Liu, Y. Chen, J. Lu, and D. Liu, "A flexible and generic gaussian sampler with power side-channel countermeasures for quantum-secure internet of things," *IEEE Internet of Things Journal*, 2020.

[26] A. P. Fournaris, C. Dimopoulos, and O. Koufopavlou, "Profiling dilithium digital signature traces for correlation differential side channel attacks," in *International Conference on Embedded Computer Systems*. Springer, 2020, pp. 281–294.

[27] 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.*, vol. 2018, p. 821, 2018.

[28] W.-L. Huang, J.-P. Chen, and B.-Y. Yang, "Power analysis on ntru prime," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 123–151, 2020.

[29] A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer, and M. Orshansky, "Horizontal side-channel vulnerabilities of post-quantum key exchange protocols," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 81–88.

[30] J. W. Bos, S. Friedberger, M. Martinoli, E. Oswald, and M. Stam, "Assessing the feasibility of single trace power analysis of frodo," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 216–234.

[31] B.-Y. Sim, J. Kwon, J. Lee, I.-J. Kim, T. Lee, J. Han, H. Yoon, J. Cho, and D.-G. Han, "Single-trace attacks on the message encoding of lattice-based kems," Cryptology ePrint Archive, Report 2020/992, 2020, https://eprint.iacr.org/2020/992.

[32] Z. Xu, O. Pemberton, S. S. Roy, and D. Oswald, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber," Cryptology ePrint Archive, Report 2020/912, 2020. https://eprint. iacr. org . . . , Tech. Rep., 2020.

[33] D. Amiet, A. Curiger, L. Leuenberger, and P. Zbinden, "Defeating newhope with a single trace," in *International Conference on Post-Quantum Cryptography*. Springer, 2020, pp. 189–205.

[34] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on cca-secure lattice-based pke and kems," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 307–335, 2020.

[35] R. Fisher, F. Yates *et al.*, "Statistical tables for biological, agricultural and medical research." *Statistical tables for biological, agricultural and medical research.*, no. 6th ed, 1963.

[36] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "NTRU Prime: round 2," Submission to the NIST PQC standardization process, URL: https://ntruprime.cr.yp.to/, 2019.

[37] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015.

[39] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[40] Bernstein, D.J., Lange, T. (eds.), "eBACS: ECRYPT Benchmarking of Cryptographic Systems," http://bench.cr.yp.to, accessed May 5, 2021.

[41] P. Kashyap, F. Aydin, S. Potluri, P. Franzon, and A. Aysu, "2deep: Enhancing side-channel attacks on lattice-based key-exchange via 2d deep learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.

[42] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 148–179, 2019.