Low-Cost Shuffling Countermeasures Against Side-Channel Attacks for NTT-Based Post-Quantum Cryptography

Zhaohui Chen, Yuan Ma, and Jiwu Jing, *Member, IEEE*

***Abstract*—Lattice-based cryptography (LBC) schemes are promising candidates in the post-quantum cryptography (PQC) standardization process. Number theoretic transform (NTT), as a crucial technique, is widely used to accelerate LBC implementations on computer systems. However, existing side-channel attacks can recover the secret key in real-world cryptographic devices bypassing mathematical problems. The motivation of this work is to provide a low-cost security-enhanced architecture for NTT-based PQC processors. We convert the nested loops in NTT to a hardware-friendly single-level loop. The corresponding architecture instantiates a unified shuffling controller to schedule the order of independent basic operations. We propose the *coefficient index randomization* and the *NTT network randomization* schemes against existing power attacks and template attacks. We further achieve high performance and efficiency on the off-the-shelf FPGAs. The shuffling schemes have a negligible impact on performance, and the resource overhead is only 9%.**

***Index Terms*—FPGA, number-theoretic transform, post-quantum cryptography (PQC), side-channel attacks (SCAs).**

I. Introduction

The quantum computing technology has brought great threats to the security of widely used RSA and elliptic curve cryptography [1]. To develop next-generation standards, the National Institute of Standards and Technology (NIST) called for quantum-resistant public-key cryptography algorithms in 2016. In the collected algorithm categories, lattice-based cryptography (LBC) has advantages in speed [2] and key size [3]. Nonetheless, physical leakages of cryptography implementations, such as timing, power consumption, and electromagnetic emission have been critical concerns.

Since the strength of profiled traces depends on the logical transitions of stored values in registers, the adversary can select an operation associated with sensitive data (e.g., the secret key) as his target. Many building blocks of LBC, e.g., number theoretic transform (NTT) [4]–[7], schoolbook-style polynomial multiplier [8], and Fujisaki-Okamoto (FO) transformation [9] are vulnerable in practice. Among the building blocks, NTT is a fundamental and universal module embedded in NewHope, CRYSTALS-Kyber, CRYSTALS-Dilithium, etc. Moreover, the secret key is input data of NTT-related operations. Thus, these operations are ideal and threatening targets of side-channel attacks (SCAs). Primas *et al.* proposed a series of template attacks (TAs), which use the power templates of modular multiplication and regular factor graph of forward NTT (FNTT) or inverse NTT (INTT) process [4], [5]. Xu *et al.* [6] constructed chosen ciphertexts as the input of decryption devices to magnify the leakage. The proposed simple power attack (SPA) targets the last stage of INTT and builds classifiers achieving over 99.6% success rate. The correlation power attacks (CPAs) in [7] avoids the precondition of profiling templates or constructing specific input. It can efficiently extract the secret key in the NTT domain using the leakage of coefficient-wise modular multiplication. These NTT-oriented attacks impute the leakage characteristics of the arithmetic operators. The adversary can utilize these leakages if the power trace is aligned with the polynomial coefficients in the time dimension.

A well-studied SCA countermeasure is to decouple the signal strength from the secret value by masking it with a random value. However, masking leads to 5.7× clock cycles on software [10] and fails to resist TAs [4] or high-order attacks. Moreover, a time-based countermeasure category called shuffling permutes the order of independent operations [11]. It destroys the alignment of power traces and avoids additional computation loads. An existing shuffling implementation leads to over 2.2× circuit resources overhead compared with the unprotected implementation [12].

This work provides shuffling countermeasures on polynomial arithmetic operators for NTT-based implementations. We first propose a single-level NTT algorithm to simplify the control logic and serialize the basic operations. Then shuffling countermeasures are proposed with 2^{135} permutation space on 256-dimensional (256-D) polynomial. We further develop a hardware architecture with a unified shuffling controller supporting both coefficient-wise operations and NTT. This work also provides a good tradeoff between security enhancement and hardware efficiency. The experimental results show that our implementations have similar performance to the unprotected baseline with only 9% additional area consumption.

II. Preliminaries

Polynomial ring is a mathematical structure represented as $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, in which n satisfies power of two and q is a prime. A polynomial $\mathbf{a} \in R_q$ can be written as $(\mathbf{a}[0], \mathbf{a}[1], \dots, \mathbf{a}[i], \dots, \mathbf{a}[n-1])$, $\mathbf{a}[i] \in \mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. The centered binomial distribution is defined as β_η for a positive integer η. In R_q, polynomial multiplication in normal domain like $\mathbf{c} = \mathbf{ab}$ is equivalent to coefficient-wise multiplication in NTT domain like $\hat{\mathbf{c}} = \hat{\mathbf{a}} \circ \hat{\mathbf{b}}$. FNTT transforms a normal polynomial to NTT domain as $\hat{\mathbf{a}}$ =FNTT($\mathbf{a}$), while INTT is the inverse transformation as $\mathbf{a}$ =INTT($\hat{\mathbf{a}}$). In this article, NTT also refers to both FNTT and INTT when we introduce their common features.

The post-quantum cryptography (PQC) third-round candidates CRYSTALS-Kyber and CRYSTALS-Dilithium are based on a variant of ring learning with errors (RLWEs) which fixes n as 256 and configures the security parameters with a dimension parameter k for polynomial vector R_q^k. For example, k of CRYSTALS-Kyber is configured as 2, 3, and 4 corresponding to security levels of AES-128,

Manuscript received 1 December 2021; revised 30 January 2022 and 31 March 2022; accepted 1 May 2022. Date of publication 10 May 2022; date of current version 22 December 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1806205; in part by the Natural Science Foundation of China under Grant 61872357; and in part by the Research Program of BJCA under Grant BJCA2020-YF0300. This article was recommended by Associate Editor S. Ghosh. *(Corresponding author: Yuan Ma.)*

Zhaohui Chen is with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: chenzhaohui17@mails.ucas.ac.cn).

Yuan Ma is with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: mayuan@iie.ac.cn).

Jiwu Jing is with the School of Cryptography, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: jwjing@ucas.ac.cn).

Digital Object Identifier 10.1109/TCAD.2022.3174142

Algorithm 1: Iterative INTT

```
Input: Polynomial $\hat{\mathbf{a}} \in R_q$ stored in $\hat{\mathbf{a}}[\ ]$, parameters $\psi^{-i}$, called
       twiddle factors, and $n^{-1} \cdot \psi^{-i}$ with $i \in [0, n)$ stored in tf.
Output: Polynomial a = INTT($\hat{a}$)
1  $\hat{\mathbf{a}} \leftarrow$ Bit-Reverse ($\hat{\mathbf{a}}$)
2  $m \leftarrow 2$
3  while $m \leq n$ do
4      $s \leftarrow 2$
5      while $s < n$ do
6          for $i \leftarrow 0$ to $m/2 - 1$ do
7              $x \leftarrow s + i$
8              $y \leftarrow s + i + m/2$
9              $z \leftarrow -2i \cdot n/m \bmod n$
10             $u \leftarrow \hat{\mathbf{a}}[x]$
11             $v \leftarrow \hat{\mathbf{a}}[y]$
12             $\omega \leftarrow$ tf[$z$]
13             $\mathbf{a}[x] \leftarrow u + v \cdot \omega \bmod q$
14             $\mathbf{a}[y] \leftarrow u - v \cdot \omega \bmod q$
15         end
16         $s \leftarrow s + m$
17     end
18     $m \leftarrow m \cdot 2$
19 end
20 for $i \leftarrow 0$ to $n - 1$ do
21     $\mathbf{a}[i] \leftarrow$ tf[$n + i$] $\cdot\ \hat{\mathbf{a}}[i] \bmod q$
22 end
23 return a
```

AES-192, and AES-256, respectively. The framework of an RLWE-based public key encryption algorithm [13], including key generation, encryption, and decryption are shown as follows.

1) *KeyGen():* Sample a uniform random polynomial $\hat{\mathbf{a}} \in R_q$. Choose two polynomial $\mathbf{s}, \mathbf{e_0}$ from β_η and compute $\hat{\mathbf{t}} = \hat{\mathbf{a}} \circ \mathrm{FNTT}(\mathbf{s}) + \mathrm{FNTT}(\mathbf{e}_0)$. The public key is $(\hat{\mathbf{a}}, \hat{\mathbf{t}})$, and the private key is $\hat{\mathbf{s}}$.
2) *Enc($\hat{\mathbf{a}}, \hat{\mathbf{t}}, \mathbf{m}$):* The sender encodes the message m to polynomial $\overline{\mathbf{m}}$ with an encoder designed to tolerate introduced errors by mapping "0" bits to 0 and "1" bits to $\lceil q/2 \rfloor$. Sample polynomial $\mathbf{r}, \mathbf{e}_1$, and $\mathbf{e}_2$ from β_η. The ciphertext consists of $\hat{\mathbf{c}}_1 = \hat{\mathbf{a}} \circ \mathrm{FNTT}(\mathbf{r}) + \mathrm{FNTT}(\mathbf{e}_1)$ and $\mathbf{c}_2 = \mathrm{INTT}(\hat{\mathbf{t}} \circ \mathrm{FNTT}(\mathbf{r})) + \mathbf{e}_2 + \overline{\mathbf{m}}$.
3) *Dec($\hat{\mathbf{s}}, \hat{\mathbf{c}}_1, \mathbf{c}_2$):* The recipient computes $\mathbf{m}' = \mathbf{c_2} - \mathrm{INTT}\ (\hat{\mathbf{s}} \circ \hat{\mathbf{c}}_1)$ and recovers the original message m from polynomial $\mathbf{m}'$ using a decoder. The decoder is used to decode a coefficient to "1" bit if the coefficient of $\mathbf{m}'$ is closer to $\lceil q/2 \rfloor$ than to 0 and vice versa.

To protect the secret key against SCAs, we focus on the decryption process, especially key-related polynomial arithmetic processes, i.e., coefficient-wise operations and INTT.

III. Proposed Hardware Architecture and Low-Cost Countermeasures

In this section, we propose a hardware-friendly NTT to serialize the access pattern of coefficients. Then, an efficient shuffling architecture is developed.

A. *Efficient Baseline Implementation*

As shown in Algorithm 1, an iterative INTT consists of three-level nested loops and a coefficient-wise multiplicative scaling. The nested loops introduce heavy logic overhead and causes propagation delay for a resource-constrained implementation [14]. We observe the following two laws on Algorithm 1: 1) each iteration fetches coefficients from different addresses and performs a fixed arithmetic logic, i.e., butterfly operation as lines 13 and 14 and 2) the indexes (x, y, z) of operands are independent with the polynomial $\hat{\mathbf{a}}$. An optimization idea is to convert the nested loops to a single-level loop. As Algorithm 2 shows, the addresses of polynomial coefficients and twiddle factors can be precomputed and stored in order. At run-time, Algorithm 2 sequentially accesses the address memory other than generating addresses on-the-fly.

Algorithm 2: Single-Level INTT

```
Input: Polynomial $\hat{\mathbf{a}} \in R_q$, parameters $\psi^{-i}$ and $n^{-1} \cdot \psi^{-i}$ with
       $i \in [0, n)$ stored in tf, and pre-computed addresses of
       coefficients and twiddle factors stored in u_addr, v_addr and
       tf_addr, respectively.
Output: Polynomial $\mathbf{a}$ = INTT($\hat{\mathbf{a}}$)
1  for $i \leftarrow 0$ to $n/2 \times logn - 1$ do
2      $u \leftarrow \hat{\mathbf{a}}$[u_addr[$i$]]             /* x is in u_addr */
3      $v \leftarrow \hat{\mathbf{a}}$[v_addr[$i$]]             /* y is in v_addr */
4      $\omega \leftarrow$ tf[tf_addr[$i$]]          /* z is in tf_addr */
5      $\hat{\mathbf{a}}$[u_addr[$i$]] $\leftarrow u + v \cdot \omega \bmod q$
6      $\hat{\mathbf{a}}$[v_addr[$i$]] $\leftarrow u - v \cdot \omega \bmod q$
7  end
8  for $i \leftarrow 0$ to $n - 1$ do
9      $\mathbf{a}[i] \leftarrow$ tf[tf_addr[$n + i$]] $\cdot\ \hat{\mathbf{a}}[i]$ mod$q$
10 end
11 return a
```

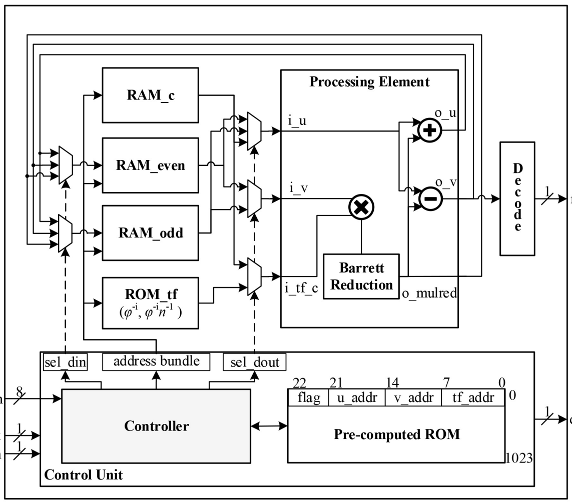

Fig. 1. Overall architecture of the proposed efficient NTT-based decryption module. For the decryption processor with shuffling countermeasures, the controller module is replaced with the shuffling controller module.

An architecture for the decryption module based on the single-level INTT is shown in Fig. 1. The architecture consists of a control unit, a processing element (PE), and memory blocks. The ciphertext $\mathbf{c}_2$ is stored in RAM_c and twiddle factors are stored in ROM_tf. We apply efficient design methodologies as follows.

Use Two-Bank RAMs to Improve Bandwidth: A butterfly operation reads and writes two coefficients of $\hat{\mathbf{c}_1}$, respectively, in a clock cycle, while each RAM block on FPGAs has only two ports. To achieve enough bandwidth, this work refers to the memory arrangement scheme in [15] and stores polynomial $\hat{\mathbf{c}}_1$ into an even bank and an odd bank according to the number of "1" bits in the binary representation of coefficient index. For an 8-dimensional (8-D) polynomial, the coefficient 0(000), 3(011), 5(101), and 6(110) are stored in RAM_even, while 1(001), 2(010), 4(100), and 7(111) are stored in RAM_odd. In the NTT computing process, the PE always accesses the four ports of RAM_even and RAM_odd without any conflict.

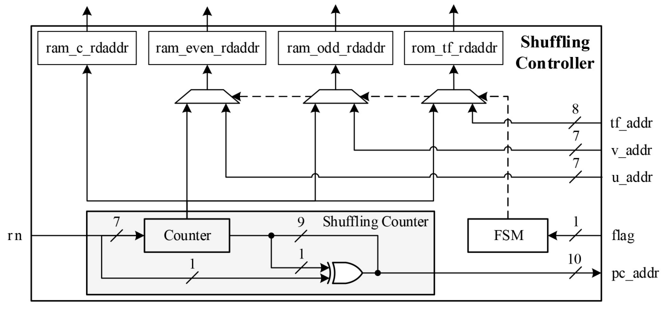

Fig. 2. Architecture of the shuffling controller for 256-D polynomials.

Simplify the Control Unit With a Precomputed ROM: As shown in Algorithm 2, the single-level INTT fetches the coefficient addresses under the control of a counter. This work stores the precomputed addresses sequentially in an $n/2 \times logn$-depth ROM, including the address for u, v, ω, and a flag bit to switch between even and odd banks. To avoid leaking the accessed precomputed indexes, padding bits are appended to the 23 significant bits to blind the power strength. The address generation logic is implemented with a counter and the precomputed ROM. Moreover, for multiple-core implementations, the overhead of the ROM can be amortized.

Integrate Basic Operations in the Datapath: We observe that the decryption process includes a polynomial subtraction operation after INTT. This work overlap its latency with line 9 of Algorithm 2. The PE uses Barrett reduction to support efficient modular arithmetic over $\mathbb{Z}_q$ and Cooley–Tukey butterfly. Since this architecture decouples the arithmetic logic unit and the control unit, other configurable butterfly units in [16] and [17] are also adaptive.

B. Shuffling Countermeasures

This section prevents the adversary from recovering the secret key by reordering the basic arithmetic operations. We propose coefficient index randomization (CIR) and NTT network randomization (NNR) against SCAs targeting point-wise operations and NTT.

CIR: A simple shuffling is to randomize the start index of a series of operations. However, this strategy has been proved invalid because it only has 16 out of 16! $\approx 2^{44}$ possible permutations for a 16-round AES implementation [11]. Even though shuffling with a full permutation provides the best security characteristics, it costs more random numbers (RNs) and logic gates. Considering the short board effect, for a 256-D polynomial, the full permutation space 256! $\approx 2^{1684}$ is over-designed for resource-constrained chips.

The shuffling controller in Fig. 2 instantiates a shuffling counter to manipulate the execution order of the sequence. RNs are input from an external true RN generator (TRNG). The CIR divides a n-round operation into $n/2$ couples and provides $2^{n/2+log(n/2)}$ permutations. The bits rn[7:1] select the first couple to be fetched and the internal order of a couple is controlled by random bit stream from rn[0]. The following example shows CIR on an 8-round operation on **a**[0] to **a**[7].

1) For rn[2:1] = “00,” rn[0] = {“1”, ‘ 0”, ‘ 1”, “1”}, the executing order is {**a**[1],**a**[0], **a**[2], **a**[3], **a**[5], **a**[4], **a**[7], **a**[6]}.
2) For rn[2:1] = “11,” rn[0] = {“1”, “0”, “0”, “1”}, the executing order is {**a**[7],**a**[6], **a**[0], **a**[1], **a**[2], **a**[3], **a**[5], **a**[4]}.

NNR: Fig. 3(a) shows the datapath of INTT on an 8-D polynomial $\hat{\mathbf{a}}$. The computation process has three stages separated into dashed boxes. Each stage of the network has four butterfly operations presented as black solid circles with two input coefficients. The indexes 0, 1, 2, and 3 in the units indicate its executing order

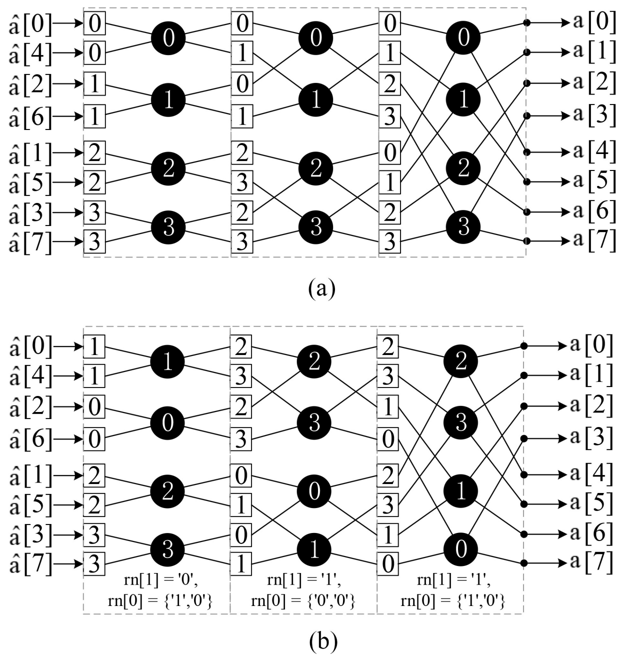
â[0]
â[4]
â[2]
â[6]
â[1]
â[5]
â[3]
â[7]
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
(a)
rn[1] = '0', rn[0] = {'1','0'}
rn[1] = '1', rn[0] = {'0','0'}
rn[1] = '1', rn[0] = {'1','0'}
(b)

Fig. 3. Datapath for an 8-D INTT. Squares represent intermediate values. Black solid circles represent butterfly operations. (a) Unprotected INTT network. (b) Shuffled INTT network.

on PE. Since the INTT network has a fixed pattern, the adversary can construct a factor graph to combine side-channel leakage. We use random bits to shuffle the order of butterflies in each stage as Fig. 3(b). The shuffling logic of each stage is similar to that of CIR. Throughout the INTT process, this shuffling scheme totally introduces $(n/4 + log(n/4)) \cdot logn$ random bits for an n-dimensional polynomial. If the adversary builds the factor graph of Fig. 3(b) as regular order, he will get a scrambled INTT network. As shown in Fig. 2, benefiting from the single-level INTT, NNR uses the pc_addr signal to manipulate the order of reading addresses of precomputed ROM with little overhead.

IV. Results and Comparisons

This section compares our low-cost SCA-resistant architecture with the state-of-the-art works in terms of performance and area costs. We also evaluate security enhancements under real-world attacks.

A. Comparisons on Performance and Area

We implemented the baseline and shuffling architectures on a Xilinx 28 nm Artix-7 FPGA. The implementation results of this work (TW) are compared with former works in Table I. The result excludes the area of TRNG since its throughput requirement is application-dependent. For example, The ES-TRNG with 1.15 Mb/s throughput proposed in [21] consumes ten LUTS and ten FFs. For a fair comparison, Table I counts latency according to decryption, i.e., the sum of INTT and 2× coefficient-wise operations. TW targets polynomial ring $\mathbb{Z}_{7681}[x]/\left\langle x^{256}+1\right\rangle$ [22] with $k = 1$. Without loss of generality, scalability is easy to achieve by repeatedly executing the basic operations polynomial-by-polynomial. Since some implementations do not disclose decryption latency [16]–[19], we count according to the same clock cycles as TW, i.e., 1548. We also present modulus parameter q in Table I, which are all between 12 and 14 bits. TW achieves reasonably high frequency benefiting from the pipeline datapath and simplified NTT controller. Our baseline shows the best efficiency according to the LUT-based area-time

TABLE I
COMPARISON OF COUNTERMEASURES AGAINST SCAs FOR NTT-BASED IMPLEMENTATIONS (**TW** = THIS WORK)

Countermeasure	FPGA	(n, q)	Freq. (MHz)	Latency(μs)	Area (LUT/FF/DSP/BRAM)	AT Product[c]	PA/TA Protection
Unprotected **TW**	Artix-7	$(256, 7681)$	370	4.2	291/312/1/3	1.0	□/□
Shuffling **TW**	Artix-7	$(256, 7681)$	370	4.2(+0%)	316(+9%)/325(+3%)/1/3	1.1	■/■
Unprotected[a] [16]	Zynq-7000	$(512, 12289)$	245	6.3	370/165/1/2.5	1.9	□/□
Unprotected[b] [17]	Artix-7	$(256, 7681)$	246	6.3	479/472/1/2	2.5	□/□
Unprotected[a] [18]	Artix-7	$(256, 3329)$	161	9.6	323/250/1/NA	2.6	□/□
Unprotected[b] [19]	Artix-7	$(256, 7681)$	176	8.8	656/555/3/2.5	4.7	□/□
Unprotected [12]	Artix-7	$(256, 7681)$	303	7.8	1163/NA/2/3	7.5	□/□
Masking [12]	Artix-7	$(256, 7681)$	250	10.1(+29%)	4269(+256%)/NA/5/6	35.5	■/□
Perm. shuffling [12]	Artix-7	$(256, 7681)$	222	11.4(+46%)	7385(+559%)/NA/2/4	69.2	■/■
LFSR shuffling [12]	Artix-7	$(256, 7681)$	277	10.3(+32%)	2861(+122%)/NA/2/3	24.2	■/■
Unprotected [20]	Virtex-II	$(256, 7681)$	120	23.5	1713/830/1/NA	33.1	□/□
Masking [20]	Virtex-II	$(256, 7681)$	100	75.2(+220%)	2014(+18%)/959(+16%)/1/NA	124.5	■/□

[a] The implementation instantiates two parallel butterfly units. We provide area by dividing the consumption of the NTT module by 2.
[b] We only count the area of an NTT core.
[c] AT Product is normalized according to Unprotected **TW**.

(AT) product. Since the proposed shuffling countermeasures do not cost additional clock cycles or introduce critical paths, shuffling TW achieves similar performance to the baseline, and loses only 10% efficiency.

TW occupies one more 36-kb block RAM than [17] and significantly improves efficiency. Further optimization can amortize this additional precomputed ROM for parallel cores. Compared with [16] and [18], the advantages mainly come from the hardware-friendly INTT scheme which avoids address generation logic and improves frequency. Compared with the extensive core [19], TW saves two DSPs on the modular multiplication module and achieves a higher frequency.

The works in [12] take the high-level synthesis (HLS) generated circuit as the baseline. However, HLS implementations are typically inferior to handcrafted code in performance and area. Although it is the most advanced HLS implementation, the efficiency of our manual code achieves 7.5× better efficiency. The overhead of our shuffling countermeasures is much less than that of the two schemes proposed in [12]. The LFSR-based shuffling scheme is biased and it only introduces 71-bit randomness. The permutation-based scheme needs a complex permutation network, and it consumes over 5.5× area occupation. Masking is another popular countermeasure, but this scheme loses efficacy with regard to single trace TAs. The masking scheme in [20] needs multiple decoding attempts and leads to high latency.

B. Evaluation on SCA Resistance

As mentioned in [4], masking countermeasures cannot resist single trace TAs. The authors emphasize that shuffling is an effective countermeasure since the factor graph becomes unpredictable. The proposed NNR scheme restrains TAs by introducing 2^{560} permutation space throughout NTT stages. The adversary can hardly align the position of butterfly operations with a correct time slot anymore.

Regarding the SPA [6], the adversary needs to combine the results of two constructed classifiers. This work shuffles the butterfly operations within each NTT stage so that a successful attack needs to distinguish from the 2^{140} permutation space.

We tried to attack the proposed baseline and shuffling architecture on FPGA, respectively. We choose the output of Barrett modular multiplication as a point of interest and execute CPA [7] to recover key coefficients with the Hamming distance model. For the unprotected

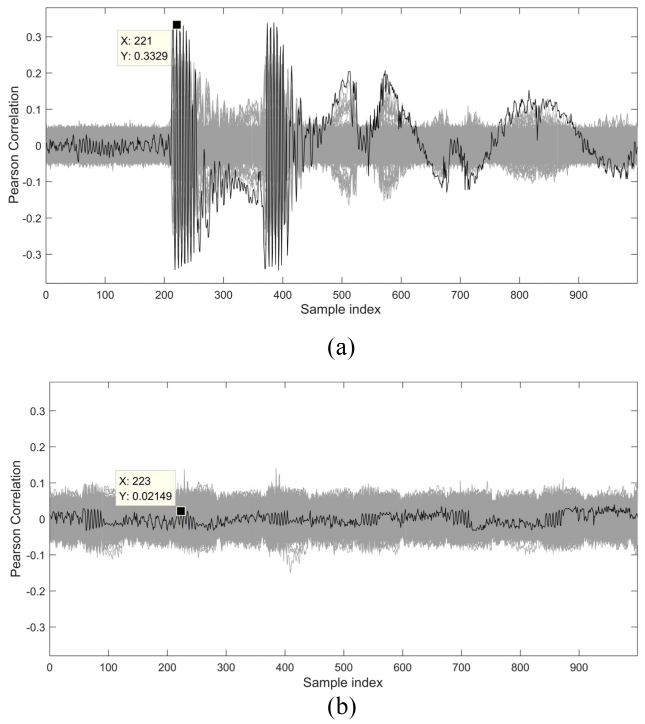

Fig. 4. CPA results on the decryption processors targeting point-wise multiplication. The black plot corresponds to the correct key. (a) Attacking the unprotected processor. (b) Attacking the processor with shuffling countermeasures.

implementation, the correlation curve shows peaks with 5000 traces as shown in Fig. 4(a). The protected implementation reduces the calculated Pearson correlation by over 10× as shown in Fig. 4(b), so that the correct key is indistinguishable from other key guesses. The adversary needs to capture $256^2\times$ traces on real-world devices (signal-to-noise ratio $\leq$ 0.2) according to the rule of thumb [23]. This is an adequate enhancement because cryptographic devices can restrict secret key usage before key refreshments. The adversary may also try to restore the order to offset shuffling, while it is infeasible for the 2^{135} permutation space under this parameter setup.

V. Conclusion

This work proposes a low-cost security-enhanced architecture for NTT-based PQC algorithms. The developed hardware architecture enables shuffling both coefficient-wise operations and NTT with a unified shuffling controller. The proposed shuffling schemes, i.e., CIR and NNR resist existing SCAs on the secret key. The experiment result shows the impact of shuffling on the performance is negligible, and its overhead on area is only 9%. To build an end-to-end SCA-resistant implementation, countermeasures for other modules (e.g., the Gaussian sampler and FO transformation) need to be applied.

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. FOCS*, Santa Fe, NM, USA, 1994, pp. 124–134.

[2] H. Nejatollahi, N. D. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–41, 2019.

[3] J. W. Bos *et al.*, "Frodo: Take off the ring! practical, quantum-secure key exchange from LWE," in *Proc. CCS*, Vienna, Austria, 2016, pp. 1006–1018.

[4] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Proc. CHES*, Taipei, Taiwan, 2017, pp. 513–533.

[5] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *Proc. LATINCRYPT*, Santiago Metropolitan Region, Chile, 2019, pp. 130–149.

[6] Z. Xu, O. M. Pemberton, S. S. Roy, D. Oswald, W. Yao, and Z. Zheng, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber," *IEEE Trans. Comput.*, early access, Oct. 27, 2021, doi: 10.1109/TC.2021.3122997.

[7] Z. Chen, E. Karabulut, A. Aysu, Y. Ma, and J. Jing, "An efficient non-profiled side-channel attack on the CRYSTALS-Dilithium post-quantum signature," in *Proc. ICCD*, Storrs, CT, USA, 2021, pp. 583–590.

[8] A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer, and M. Orshansky, "Horizontal side-channel vulnerabilities of post-quantum key exchange protocols," in *Proc. HOST*, Washington, DC, USA, 2018, pp. 81–88.

[9] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 3, pp. 307–335, 2020.

[10] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical CCA2-secure and masked Ring-LWE implementation," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 1, pp. 142–174, 2018.

[11] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert, "Shuffling against side-channel attacks: A comprehensive study with cautionary note," in *Proc. ASIACRYPT*, Beijing, China, 2012, pp. 740–757.

[12] T. Zijlstra, K. Bigou, and A. Tisserand, "FPGA implementation and comparison of protections against SCAs for RLWE," in *Proc. INDOCRYPT*, Hyderabad, India, 2019, pp. 535–555.

[13] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange—A new hope," in *Proc. 25th USENIX Security*, Austin, TX, USA, 2016, pp. 327–343.

[14] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "Towards efficient kyber on FPGAs: A processor for vector of polynomials," in *Proc. ASP-DAC*, Beijing, China, 2020, pp. 247–252.

[15] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Proc. LATINCRYPT*, Santiago, Chile, 2012, pp. 139–158.

[16] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 2, pp. 49–72, 2020.

[17] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "High-performance area-efficient polynomial ring processor for CRYSTALS-Kyber on FPGAs," *Integration*, vol. 78, pp. 25–35, May 2021.

[18] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.

[19] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, early access, Aug. 19, 2020, doi: 10.1109/TC.2020.3017930.

[20] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked Ring-LWE implementation," in *Proc. CHES*, Saint-Malo, France, 2015, pp. 683–702.

[21] B. Yang, V. Rozic, M. Grujic, N. Mentens, and I. Verbauwhede, "ES-TRNG: A high-throughput, low-area true random number generator based on edge sampling," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 267–292, 2018.

[22] J. W. Bos *et al.*, "CRYSTALS—Kyber: A CCA-secure module-lattice-based KEM," in *Proc. EuroS&P*, London, U.K., 2015, pp. 353–367.

[23] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards* (Advances in Information Security). New York, NY, USA: Springer-Verlag, 2007, pp. 147–148.