IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 26 November 2023, accepted 8 December 2023, date of publication 14 December 2023, date of current version 21 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3342914

RESEARCH ARTICLE

Systematization of Shuffling Countermeasures: With an Application to CRYSTALS-Dilithium

JONGHYEOK LEE [1], **JAESEUNG HAN** [1], **SANGYUB LEE** [2], **JIHOON KWON** [2], **KEON-HEE CHOI** [1], **JAE-WON HUH** [1], **JIHOON CHO** [2], **AND DONG-GUK HAN** [1]
[1]Department of Financial Information Security, Kookmin University, Seoul 02707, Republic of Korea
[2]Security Research Team, Samsung SDS Inc., Seoul 06765, Republic of Korea

Corresponding author: Dong-Guk Han (christa@kookmin.ac.kr)

This work was supported in part by the Military Crypto Research Center funded by the Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD) under Grant UD210027XD; and in part by the Ministry of Science and ICT (MSIT), South Korea, under the Information Technology Research Center (ITRC) Support Program Supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) under Grant IITP-2023-RS-2022-00164800.

ABSTRACT Shuffling is an essential countermeasure employed during the implementation of cryptographic algorithms to mitigate vulnerabilities against side-channel attacks, regardless of the algorithm's nature. However, a comprehensive and structured shuffling framework has yet to be established, resulting in the need for developers to create customized solutions adapted to their specific algorithmic or operational requirements. This research paper introduces an innovative and systematic shuffling framework, providing developers with a set of guidelines to effectively select suitable shuffling methodologies aligned with their specific objectives. Additionally, we illustrate the application of this framework to the CRYSTALS-Dilithium signature algorithm, a finalist in NIST's Post-Quantum Cryptography (PQC) standardization process. By leveraging our framework, we devise shuffling countermeasures and present an extensive array of twelve shuffling schemes. For each scheme, shuffling schemes are applied universally to all operations involving any confidential data, regardless of existence of known attacks targeting corresponding data. We also measured the performance of implementations of our shuffling schemes, the minimal overhead is 12.4%.

INDEX TERMS Side-channel countermeasure, hiding countermeasure, shuffling countermeasure, horizontal shuffling, vertical shuffling.

I. INTRODUCTION

The security of widely used public key cryptosystems such as RSA and ECC is based on mathematical hard problems that factoring large integer and solving the discrete logarithm problem. In 1994, Shor proposed a quantum algorithm that can effectively break these hard problems [1]. Moreover, the recent significant advancements in quantum computers have increased the threat to modern cryptosystems. As a result, the research and development of post-quantum cryptography (PQC) (also known as quantum-resistant cryptography) have been triggered.

In 2016, National Institute of Standards and Technology (NIST) initiated PQC standardization project to develop additional public key cryptography including key encapsulation mechanism and digital signature algorithms [2]. The design criteria of PQC algorithms are to ensure the security and efficiency of cryptographic algorithms which are secure against both classical and quantum attacks. After multiple rounds, in 2022, NIST announced four algorithms for standardization, and fourth round candidate algorithms [3]. NIST also recommended two primary algorithms for most use cases: CRYSTALS-Kyber for key establishment and CRYSTALS-Dilithium for digital signatures [4], [5].

Side-channel attack (SCA) is a physical attack utilizing side-channel information, such as timing variations, power consumption, or electromagnetic radiation, leaked during

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

execution of cryptographic algorithms [6]. Although the design of PQC algorithms is focused on protecting against quantum attacks, their implementations may still be vulnerable to SCAs. Therefore, during the NIST PQC standardization project, NIST wanted to gather more information about the costs of implementing in a way that provides resistance to SCAs [7].

To mitigate SCAs, there are two common countermeasures called masking and hiding schemes [8]. Masking scheme aims to randomize the intermediate data with additional data, called masks, during the execution. On the other hand, hiding scheme aims to make the side-channel leakage independent of intermediate values or at least reduce the linkage between them. Shuffling is one of the most common one among hiding schemes and it can be the better choice for an implementation against SCAs because of the heavy cost of masking when deployed carefully.

There are a generalized and systematic framework for masking schemes like d-th order masking and ISW masking schemes [9]. However, to the best of our knowledge, hiding countermeasures have only been proposed in an algorithm-specific manner. For instance, [10] proposed a shuffling scheme specialized for AES when implemented on smart cards, and [11], [12] proposed shuffling schemes tailored for long integer multiplication and number theoretic transform (NTT) used in various cryptographic algorithms. Additionally, [13], [14], [15] presented shuffling schemes specifically designed for Saber and CRYSTALS-Dilithium, which are currently undergoing standardization in NIST's PQC competition. Although various shuffling schemes specialized for different algorithms have been extensively proposed, a generalized and systematic framework, similar to what exists for masking schemes, has not been put forward.

Such a structured and generalised framework helps countermeasure designers to facilitate the design of shuffling schemes. They can identify the appropriate shuffling characteristics for their targeted cryptographic algorithms. By identifying the potential dimensions and dependencies of the permutations used in the horizontal shuffling, as well as the number of folds in the potential vertical shuffling, they can determine the possible shuffling methods by their combinations. Consequently, designers can select the optimal shuffling method from among the feasible options, taking into account the appropriate trade-off between security and performance within their target implementation environment.

A. OUR CONTRIBUTIONS

We make a dual contribution in this study. Firstly, we introduce a novel and comprehensive framework for shuffling, addressing a significant gap in the existing literature. Unlike previous research that focused on specific cryptographic algorithms or customized techniques for particular operations [10], [11], [12], [13], [14], [15], our framework offers a general approach akin to masking. By providing implementers with clear guidelines on incorporating shuffling into their target cryptographic algorithms, we alleviate the challenges of considering all relevant factors without proper guidance.

Secondly, we apply our framework to the CRYSTALS-Dilithium signature algorithm, which has been selected as a finalist in NIST's PQC standardization process. We develop and implement twelve distinct shuffling countermeasures, taking into account both the targeted operations of previous attacks and operations involving secret values that have not yet been exploited. Through a comparative performance analysis, we demonstrate that our proposed countermeasures incur a minimal overhead of 12.4%. Overall, our contributions encompass the establishment of a systematic shuffling framework and its practical application to the CRYSTALS-Dilithium algorithm, providing valuable guidance and effective countermeasures in the domain of side-channel analysis.

B. ORGANIZATION

The structure of this paper is as follows. In Section II, we provide an explanation of shuffling countermeasures and their application to various cryptographic algorithms. Section III presents our proposed systematic framework for shuffling countermeasure. Within this framework, we categorize shuffling countermeasure into horizontal and vertical shuffling and provide a detailed explanation of their respective characteristics. Moving on to Section IV, we describe the application of the shuffling framework proposed in Section III to CRYSTALS-Dilithium. We discuss the methods of applying the framework to CRYSTALS-Dilithium and compare the performance of different applicable shuffling countermeasures. Finally, in Section V, we summarize the contributions made in this paper and conclude by discussing the results of the applied shuffling schemes.

II. RELATED WORK

The objective of the hiding scheme is to ensure that the power consumption of cryptographic devices remains unaffected by intermediate values and operations performed [8]. There are two approaches to achieving this goal. The first approach involves inducing random power consumption in devices, while the second approach aims to maintain consistent power consumption across all data values and operations. However, achieving complete randomization or uniformity of power consumption in cryptographic devices is not practically feasible. Nevertheless, there have been several proposals that aim to approximate this objective. These proposals can be categorized into two groups. The first group focuses on altering the timing of operations to introduce randomized power consumption. By modifying the execution timing of operations, the power consumption becomes less predictable and exhibits a more random pattern. The second group of proposals involves techniques that manipulate the amplitude of power consumption. Although these proposals may not achieve perfect randomization or uniformity, they offer methods to approach the goal of mitigating power analysis attacks by introducing power variations and reducing correlations between power consumption and sensitive data or operations.

A. SHUFFLING COUNTERMEASURES

Shuffling is a widely utilized technique to introduce randomness in the timing of operations, thus enhancing the security of cryptographic devices against power analysis attacks. By shuffling the order of operations, the execution timing becomes less predictable, introducing a level of randomization in power consumption. The objective is to disrupt any patterns or correlations that may exist in the power consumption profile, making it challenging for attackers to analyse and exploit the power side-channel information.

There are various methods available for shuffling operations, each with its own unique characteristics. Some commonly used techniques include:

- Random Permutation (RP): This method involves randomly permuting the order of operations using a random permutation function [16]. By applying a random permutation, the execution order of operations is randomized, enhancing the level of shuffling.
- Random Start Index (RSI): In this approach, the starting index of operations is randomized [17]. Each time the operations are executed, a different starting point is chosen, leading to varying execution orders and introducing additional variability.
- Reverse Shuffle (RS): This method allows the selection of either a forward or reverse execution order for the operations [17]. By enabling the reverse direction as an option, it introduces further randomness in the order of operations, making it more challenging to establish patterns.
- Sweep Swap Shuffle (SSS): This technique expands the operation sequence into a matrix form and performs the operations based on row or column order [17]. By reshaping the operation sequence and executing based on row or column order, it introduces randomness in the execution order.

These methods offer varying levels of shuffling and can be applied based on the specific requirements and constraints of the system or algorithm. The primary goal is to disrupt any inherent patterns or correlations, thereby enhancing the security against power analysis attacks.

B. SHUFFLING ON VARIETY CRYPTOGRAPHIC ALGORITHM

Shuffling countermeasures serve as an effective option to mitigate side-channel analysis in cryptographic algorithms, regardless of their type, including symmetric key, public key, and PQC algorithms. Herbst et al. introduced a technique for applying shuffling to software implementations of AES in 2006 [10], which was later extended by Tillich et al. [18]. Additionally, Rivain et al. proposed a method that combines higher-order masking with shuffling for enhanced security [19]. These papers specifically focus on shuffling methods for AES, as block ciphers commonly employ a one-dimensional array to represent intermediate states. Consequently, these techniques can be readily adapted to other block ciphers with similar one-dimensional state representations.

For classical public key algorithms like RSA and ECC, Lee et al. proposed a shuffling method for long integer multiplication operations [11]. Moreover, Nguyen and Pham presented a shuffling technique for operations in the Montgomery domain [20]. Since the discussion on PQC began, numerous attack papers on PQC algorithms, such as Saber and CRYSTALS-Dilithium, have emphasized the necessity of shuffling schemes to counter the proposed attacks [12], [13], [14], [15].

Similar to masking, shuffling is an indispensable countermeasure in safeguarding cryptographic algorithms against side-channel analysis, regardless of their type. In the context of attacks on PQC, profiling attacks have received significant attention, and relying solely on masking may not be sufficient to thwart such attacks. As a result, shuffling has emerged as a crucial defence mechanism and has gained considerable attention in the field.

III. SYSTEMATIZATION OF SHUFFLING

As mentioned in Section II-B, a significant number of papers have been published on the application of shuffling countermeasures. However, most of these papers propose methods that are specific to particular cryptographic algorithms, and a generalized framework for shuffling countermeasures has not been presented. Therefore, in this section, our objective is to define shuffling, classify different types of shuffling, and establish properties that can be associated with each categorized shuffling method.

Definition 1 (Shuffling): Shuffling is a technique used to randomize the order of *instructions*, which include both *functions* and *operations*, within a cryptographic algorithm.

Definition 1 provides the definition of shuffling as defined by Mangard et al. [8], where shuffling is described as a method of randomizing the order of instructions. In this study, we further classify instructions into two categories: functions and operations. Based on this classification, we categorize shuffling into two types: horizontal shuffling and vertical shuffling. The classification of functions and operations can vary depending on the perspective and context. For example, in the case of the Advanced Encryption Standard (AES), functions such as AddRoundKey, SubBytes, ShiftRows, and MixColumns can be considered. These functions represent higher-level operations that manipulate the data. Within the SubBytes function, individual Sbox operations can be viewed as lower-level operations. It is evident that a function encompasses multiple operations within its scope.

A. HORIZONTAL SHUFFLING

Definition 2 (Horizontal Shuffling): Horizontal shuffling refers to the technique of randomizing the order of *operations* within a *function* in a cryptographic algorithm. It involves

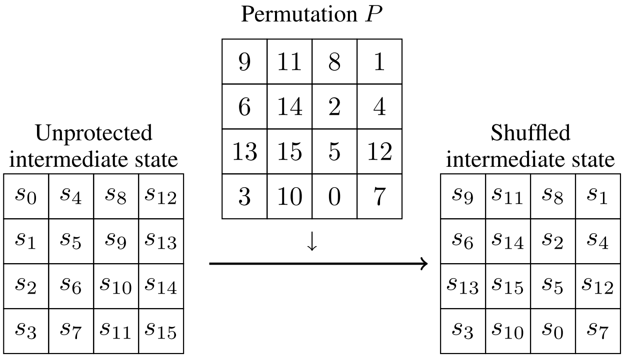


FIGURE 1. **Example of applying permutation to AES's intermediate state.**

rearranging the sequence of operations within a function to introduce randomness in their execution order.

Horizontal shuffling is defined as the process of rearranging the order of independent operations within a function to introduce randomness in the execution order, while preserving the overall functionality of the function (Definition 2). To achieve horizontal shuffling, a randomly generated permutation is commonly used, representing a random execution order for the independent operations within the function. By applying this random permutation, the execution order of operations is shuffled, leading to increased variability and reduced correlation between power consumption and sensitive data.

Property 1 (Permutation Dependency): Permutation dependency indicates the degree to which functions share permutations. All functions can share one permutation (*Full-Dependence*), or functions can be grouped and only share in a group (*Group-Dependence*). And all functions can use permutations independently (*Independence*).

Permutations used in horizontal shuffling have two properties. The first property relates to the dependency between permutations used for different functions where horizontal shuffling is applied. Based on this dependency, there are three types of permutations used in horizontal shuffling:

- Full-Dependence: In this case, a single permutation is repeated throughout the entire cryptographic algorithm. The same permutation is used repeatedly for all functions where horizontal shuffling is applied.
- Group-Dependence: Functions that undergo shuffling are grouped together, and each group shares a common permutation. However, different groups utilize independent permutations. This allows for variations in the shuffling patterns within different groups of functions.
- Independence: Each function that undergoes shuffling uses its own independent permutation. This provides the highest level of variability and randomness, as each function has a unique shuffling order.

These different levels of permutation dependency offer flexibility in the application of horizontal shuffling, enabling various trade-offs between security and overhead.

In the encryption algorithm that aims to apply shuffling countermeasure, there are l functions that use secret values, and these l functions can be grouped into g groups. In the case of Full-Dependence, a single permutation P is used to apply shuffling permutation to the l functions. Here, permutation refers to an array that stores the randomly shuffled indexes of an intermediate values or intermediate state. For example, an unprotected intermediate state of AES can be shuffled by a permutation P as depicted in Figure 1. In the case of Group-Dependence, g permutations $P_0, P_1, \cdots, P_{g-1}$ are generated, and the functions belonging to the i-th group share the same permutation P_i. This means that each group of functions uses a distinct permutation. Finally, in the case of Independence, l permutations $P_0, P_1, \ldots, P_{l-1}$ are generated, and only the i-th function uses the permutation P_i. This means that all l functions use unique permutations.

Property 2 (Permutation Dimension): Permutation dimension is the maximum number of dimensions to use when creating permutations. If the maximum operation dimension of functions is p, and the result is $b = f(a) \in \mathbb{F}^{d_1 \times d_2 \times \cdots \times d_p}$, then the permutation dimension is less than or equal to p.

The second property relates to the dimension of the permutation used in horizontal shuffling. The intermediate states targeted within the functions where shuffling is applied may not be limited to one-dimensional arrays; they can also be multidimensional arrays. For instance, if an intermediate state is a 2D array of size $K \times L$, there are two approaches for generating permutations. The first approach, the 1-D permutation approach, involves creating a one-dimensional permutation of size K and a separate one-dimensional permutation of size L. The shuffling process repeatedly uses the L-sized one-dimensional permutation whenever each element of the K-sized permutation is invoked. The second approach, the 2-D permutation approach, utilizes a one-dimensional permutation of size K and a two-dimensional permutation of size $K \times L$. With this approach, a different one-dimensional permutation of size L is chosen for each corresponding one-dimensional permutation of size K. The goal is to have K independent permutations of size L, where each permutation of size L is independent from the others.

AES's MixColumns is a suitable example. MixColumns can be represented as the equation $s' = MC(s) \in GF(2^8)^{4\times4}$. Like this, MixColumns is a 2-dimensional operation, which means the permutation dimension can be up to 2. Figure 2 (a) illustrates MixColumn without applying the permutation. The first operation to be performed is $s'_0 = (2 \bullet s_0) \oplus (3 \bullet s_1) \oplus s_2 \oplus s_3$.

An example of applying the shuffling countermeasure using 1-D permutations is shown in Figure 2 (b), where permutations P_{col} for column indices and P_{row} for row indices are generated. P_{col} is applied first, followed by the application of P_{row} to each individual column. Since a common row permutation P_{row} is used for each column, the matrix multiplication in Figure 2 (a) is rearranged to the matrix multiplication in Figure 2 (b). In this case, the first operation to be performed is to compute the value of the 0th

$$\begin{array}{|c|c|c|c|}\hline s'_0 & s'_4 & s'_8 & s'_{12} \\ \hline s'_1 & s'_5 & s'_9 & s'_{13} \\ \hline s'_2 & s'_6 & s'_{10} & s'_{14} \\ \hline s'_3 & s'_7 & s'_{11} & s'_{15} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 2 & 3 & 1 & 1 \\ \hline 1 & 2 & 3 & 1 \\ \hline 1 & 1 & 2 & 3 \\ \hline 3 & 1 & 1 & 2 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|c|}\hline s_0 & s_4 & s_8 & s_{12} \\ \hline s_1 & s_5 & s_9 & s_{13} \\ \hline s_2 & s_6 & s_{10} & s_{14} \\ \hline s_3 & s_7 & s_{11} & s_{15} \\ \hline \end{array}$$

(a) Unprotected

$$P_{col} \begin{array}{|c|c|c|c|}\hline 2 & 1 & 3 & 0 \\ \hline \end{array}$$

$$P_{row} \begin{array}{|c|c|c|c|}\hline 0 & 3 & 1 & 2 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|}\hline s'_8 & s'_4 & s'_{12} & s'_0 \\ \hline s'_{11} & s'_7 & s'_{15} & s'_3 \\ \hline s'_9 & s'_5 & s'_{13} & s'_1 \\ \hline s'_{10} & s'_6 & s'_{14} & s'_2 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 2 & 1 & 3 & 1 \\ \hline 1 & 1 & 2 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 3 & 2 & 1 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|c|}\hline s_8 & s_4 & s_{12} & s_0 \\ \hline s_{11} & s_7 & s_{15} & s_3 \\ \hline s_9 & s_5 & s_{13} & s_1 \\ \hline s_{10} & s_6 & s_{14} & s_2 \\ \hline \end{array}$$

(b) Protected using 1-D permutation

$$P_{col} \begin{array}{|c|c|c|c|}\hline 2 & 1 & 3 & 0 \\ \hline \end{array}$$

$$P_{row} \begin{array}{|c|c|c|c|}\hline 0 & 3 & 1 & 2 \\ \hline 1 & 3 & 0 & 2 \\ \hline 3 & 0 & 2 & 1 \\ \hline 1 & 2 & 0 & 3 \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline s'_{11} \\ \hline s'_8 \\ \hline s'_{10} \\ \hline s'_9 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 1 & 3 \\ \hline 1 & 1 & 3 & 2 \\ \hline 3 & 1 & 2 & 1 \\ \hline 2 & 3 & 1 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline s_{11} \\ \hline s_8 \\ \hline s_{10} \\ \hline s_9 \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline s'_5 \\ \hline s'_7 \\ \hline s'_4 \\ \hline s'_6 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 3 & 1 & 2 & 1 \\ \hline 2 & 1 & 1 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 1 & 2 & 3 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline s_5 \\ \hline s_7 \\ \hline s_4 \\ \hline s_6 \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline s'_{13} \\ \hline s'_{14} \\ \hline s'_{12} \\ \hline s'_{15} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 3 & 1 & 2 & 1 \\ \hline 2 & 3 & 1 & 1 \\ \hline 1 & 2 & 1 & 3 \\ \hline 1 & 1 & 3 & 2 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline s_{13} \\ \hline s_{14} \\ \hline s_{12} \\ \hline s_{15} \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline s'_0 \\ \hline s'_3 \\ \hline s'_1 \\ \hline s'_2 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 2 & 1 & 3 & 1 \\ \hline 1 & 1 & 2 & 3 \\ \hline 1 & 3 & 1 & 2 \\ \hline 3 & 2 & 1 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline s_0 \\ \hline s_3 \\ \hline s_1 \\ \hline s_2 \\ \hline \end{array}$$

(c) Protected using 2-D permutation

FIGURE 2. **Example of permutation dimension on AES's MixColumns.**

row of the 2nd column, which is $s'_8 = (2 \bullet s_8) \oplus s_{11} \oplus (3 \bullet s_9) \oplus s_{10}$.

Figure 2 (c) demonstrates an example of applying the shuffling countermeasure using 2-D permutations. The permutation P_{col} for column indices is the same as the 1-D permutation, but independent permutations P_{row} for row indices are generated for each column. In this case, the first operation to be performed is to compute the value of the 3rd row of the 2nd column, which is $s'_{11} = s_{11} \oplus (2 \bullet s_8) \oplus s_{10} \oplus (3 \bullet s_9)$.

These approaches provide different strategies for applying permutations to multidimensional arrays, enabling effective shuffling of the intermediate state in cryptographic algorithms. It is important to note that if the maximum dimension of the operation is p, the permissible dimension for the permutation is p or lower. As depicted in Figure 3, horizontal shuffling can be visualized in a plane where permutation dependency and permutation dimension represent the axes.

B. VERTICAL SHUFFLING

Definition 3 (Vertical Shuffling): Vertical shuffling is a method of randomizing the order of *functions* in an cryptographic algorithm. If there are multiple f_2 functions within f_1 function, multiple f_3 functions within f_2 function, and so up to f_v, then up to v-fold vertical shuffles are possible.

Vertical shuffling is a method of shuffling the order of function invocations within a cryptographic algorithm. It has two key characteristics:

- Independence: The target functions for vertical shuffling must be independent of each other. This means that their execution order can be altered without affecting the output. If the functions perform different operations, they can be distinguished by observing side-channel traces. For example, in AES, the SubBytes and ShiftRows functions can be invoked in any order without impacting the result. Although the original AES specification specifies the order of performing SubBytes followed by ShiftRows, these functions have an independent relationship, allowing for the possibility of performing ShiftRows first without any issues. Shuffling the order of executing the SubBytes and ShiftRows functions randomly is referred to as 1-fold vertical shuffling.
- Function Grouping: Functions can be grouped and composed into higher-level functions, and the order of these function groups can be shuffled, while maintaining the order of functions within each group. This is known as 2-fold vertical shuffling. By generalizing this concept, if functions are grouped into v layers and these layers can be independently shuffled, it is possible to apply v-fold vertical shuffling.

One of the most representative examples of vertical shuffling is RSA-CRT (Chinese Remainder Theorem). In RSA-CRT, the signature is generated by computing $s_p = m^{d_p} \bmod p$ and $s_q = m^{d_q} \bmod q$, where $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$. Then, $S = s_q + \left(\left(s_p - s_q \bmod p\right) \cdot \left(q^{-1} \bmod p\right) \bmod p\right) \cdot q$ is calculated. The processes of creating s_p and s_q are independent, allowing the randomization of their execution order. This is referred to as 1-fold vertical shuffling.

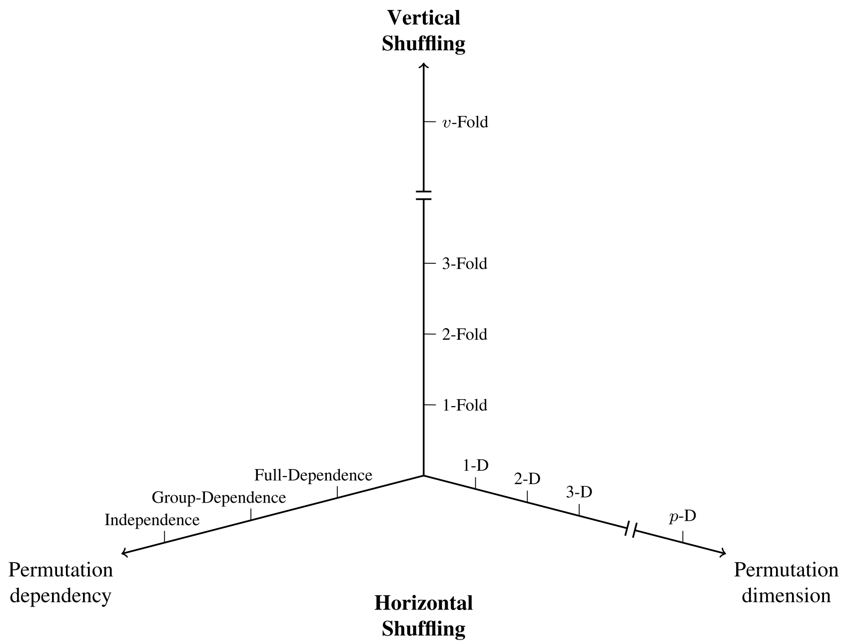


FIGURE 3. Shuffling space.

In summary, vertical shuffling allows for the randomization of function execution order, considering both single-layer and multi-layer shuffling where functions can be grouped and shuffled independently at different levels. This concept is represented by the vertical axis in Figure 3.

IV. RESULTS AND DISCUSSIONS

This section presents an example of applying the shuffling countermeasure, as defined in Section III, to CRYSTALS-Dilithium.

A. SHUFFLING ON CRYSTALS-DILITHIUM

1) CRYSTALS-DILITHIUM

CRYSTALS-Dilithium [5] is one of the three digital signature finalists of the NIST PQC standardization process. Its security is based on the hardness of the Module Learning with Errors (M-LWE) and Module Short Integer Solution (M-SIS) problems.

CRYSTALS-Dilithium consists of three different parameter sets: Dilithium2, Dilithium3, and Dilithium5, depending on NIST security levels. It is operating in the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$, where $n = 256$ and q is $8380417 = 2^{23} - 2^{13} + 1$. The remaining parameters are given in Table 1.

Algorithm 1 and Algorithm 2 specify key generation and signature generation of CRYSTALS-Dilithium, respectively. Please note that all lines of the algorithms are based on the implementation in PQClean [21]. In Algorithm 1, although not explicitly written as a function, there is a process of packing the public key pk and the secret key sk in line 13. Similarly, in Algorithm 2, there is a process of unpacking sk before line 1. The NTT representation of $a \in \mathcal{R}_q$ is denoted as $\widehat{a} = (a(r_0), a(-r_0), \ldots, a(r_{127}), a(-r_{127})) \in \mathbb{Z}_q^{256}$,

TABLE 1. Parameters of CRYSTALS-Dilithium.

NIST Security Level	2	3	5
Parameters			
q [modulus]	8380417	8380417	8380417
d [dropped bits from t]	13	13	13
τ [# of ± 1's in c]	39	49	60
challenge entropy [$\log\binom{256}{\tau} + \tau$]	192	225	257
γ_1 [y coefficient range]	2^{17}	2^{19}	2^{19}
γ_2 [low-order rounding range]	$q-1/88$	$q-1/32$	$q-1/32$
(k,l) [dimensions of A]	$(4,4)$	$(6,5)$	$(8,7)$
η [secret key range]	2	4	2
β [$\tau\cdot\eta$]	78	196	120
ω [max. # of 1's in the hint h]	80	55	75
Repetitions [$\approx e^{-256\cdot\beta(l/\gamma_1+k/\gamma_2)}$]	4.25	5.1	3.85

where $r_i = r^{brv(128+i)} \mod q$ and $r = 1753$ which is the 512-th root of unity modulo q. All multiplication operations are efficiently processed using point-wise multiplication in the NTT domain for optimization purposes. For more details, please refer to the original article [5].

2) FUNCTIONS TO BE PROTECTED

In CRYSTALS-Dilithium, the elements of the secret key sk that should not be exposed to an attacker are K, s_1, s_2, and t_0. Therefore, all operations involving these elements are potentially vulnerable to attacks.

In Algorithm 1 for key generation, an attacker may attempt to obtain s_1 or s_2 by attacking line 4 where they are generated. Lines 5 to 8 use s_1, line 9 uses both s_1 and s_2, and lines 10 and 11 use t_0, all of which are potential targets for attacks. Finally, the process of packing the sk elements

Algorithm 1 CRYSTALS-Dilithium: KeyGen

Input: -

Output: Public key $pk = (\rho, t_1)$,
Secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$

1: $\zeta \leftarrow \{0, 1\}^{256}$
2: $(\rho, \varsigma, K) \in \{0, 1\}^{256\times 3} := \mathrm{H1}\left(\zeta\right)$
3: $\widehat{A} \in \mathcal{R}_q^{k\times l} := \mathrm{ExpandA}\left(\rho\right)$
4: $(s_1, s_2) \in \mathcal{S}_\eta^l \times \mathcal{S}_\eta^k := \mathrm{H2}\left(\varsigma\right)$
5: $\widehat{s_1} := \mathrm{NTT}\left(s_1\right)$
6: $\widehat{t} := \widehat{A}\widehat{s_1}$
7: $\widehat{t} = \widehat{t} \bmod^{+} Q$
8: $t := \mathrm{NTT}^{-1}\left(\widehat{t}\right)$
9: $t = t + s_2$
10: $t = t \bmod^{\pm} Q/2$
11: $(t_1, t_0) := \mathrm{Power2Round}\left(t, d\right)$
12: $tr \in \{0, 1\}^{384} := \mathrm{CRH}\left(\rho \parallel t_1\right)$
13: **return** $pk = (\rho, t_1), sk = (\rho, K, tr, s_1, s_2, t_0)$

in line 13 can also be considered as a potential target for attacks.

In Algorithm 2 for signature generation, there are more potential points of attack compared to Algorithm 1. First, the process of unpacking sk before executing Algorithm 2 can be a target for attacks, similar to the packing process in Algorithm 1. In line 4, NTT transformations are performed on s_1, s_2, and t_0 individually. The operations from lines 5 to 11 are initiated from the secret value K. Lines 15 to 18 involve the usage of s_1. Furthermore, lines 20 to 24 involve the usage of K and s_2, while lines 25 to 28 utilize t_0. Lastly, the secret values associated with lines 29 and 30 are K, s_2, and t_0. Although line 19 is an operation associated with s_1, it is solely a range verification operation for the values included in the disclosed signature. Therefore, it does not involve any sensitive computations that require protection.

CRYSTALS-Dilithium uses SHAKE for random value extraction. In Algorithm 1, line 2, and Algorithm 2, line 2, SHAKE is utilized. In Algorithm 1, line 2, SHAKE is performed to generate ρ, ς, and K. ρ is a public value, and ς serves as the seed for generating s_1 and s_2 using rejection sampling. Even if an attacker could reveal the values of s_1 and s_2 due to the shuffling applied during rejection sampling, she wouldn't be able to determine the indices where each value is stored. Thus, applying shuffling to the part generating ς would not enhance security. Therefore, the shuffling is applied only to the part generating K. In Algorithm 2, line 2, y is generated using ρ', which is generated using SHAKE as a seed. In the deterministic version, to ensure that the same signature is generated for the same message, shuffling cannot be applied to the part generating ρ'. Countermeasures such as masking scheme or dummy operations could be used for this part, but these options lie beyond the scope of this paper and are left to the designer's discretion.

Algorithm 2 CRYSTALS-Dilithium: Sign

Input: Message m,
Secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$

Output: Signature $\sigma = (z, c, h)$

1: $\mu \in \{0, 1\}^{384} := \mathrm{CRH}\left(tr \parallel m\right)$
2: $\rho' \in \{0, 1\}^{384} := \mathrm{CRH}\left(K \parallel \mu\right)$
3: $\widehat{A} \in \mathcal{R}_q^{k\times l} := \mathrm{ExpandA}\left(\rho\right)$
4: $\kappa := 0, \widehat{s_1} := \mathrm{NTT}\left(s_1\right), \widehat{s_2} := \mathrm{NTT}\left(s_2\right)$, $\widehat{t_0} := \mathrm{NTT}\left(t_0\right)$
5: $y \in \widetilde{S_{\gamma_1}}^{l} := \mathrm{ExpandMask}\left(\rho', \kappa++\right)$
6: $\widehat{y} := \mathrm{NTT}\left(y\right)$
7: $\widehat{w} := \widehat{A}\widehat{y}$
8: $\widehat{w} = \widehat{w} \bmod^{+} Q$
9: $w := \mathrm{NTT}^{-1}\left(\widehat{w}\right)$
10: $w = w \bmod^{\pm} Q/2$
11: $w_1 := \mathrm{Decompose}\left(w, 2\gamma_2\right)$
12: $\widetilde{c} \in \{0, 1\}^{256} := \mathrm{H}\left(\mu \parallel w_1\right)$
13: $c \in B_\tau := \mathrm{SampleInBall}\left(\widetilde{c}\right)$
14: $\widehat{c} := \mathrm{NTT}\left(c\right)$
15: $\widehat{z} := \widehat{c}\widehat{s_1}$
16: $z := \mathrm{NTT}^{-1}\left(\widehat{z}\right)$
17: $z = z + y$
18: $z = z \bmod^{+} Q$
19: If $\|z\|_\infty \geq \gamma_1 - \beta$, then goto line 5
20: $\widehat{r_0} := \widehat{c}\widehat{s_2}$
21: $r_0 = \mathrm{NTT}^{-1}\left(\widehat{r_0}\right)$
22: $r_0 = w - r_0$
23: $r_0 = r_0 \bmod^{+} Q$
24: If $\|r_0\|_\infty \geq \gamma_2 - \beta$, then goto line 5
25: $\widehat{h} := \widehat{c}\widehat{t_0}$
26: $h = \mathrm{NTT}^{-1}\left(\widehat{h}\right)$
27: $r_1 := h \bmod^{+} Q$
28: If $\|r_1\|_\infty \geq \gamma_2$, then goto line 5
29: $r_0 = r_0 + r_1$
30: $h := \mathrm{MakeHint}\left(-r_1, r_0, 2\gamma_2\right)$
31: If $\|h\|_\infty > \omega$, then goto line 5
32: **return** $\sigma = (z, \widetilde{c}, h)$

3) HORIZONTAL SHUFFLING

In CRYSTALS-Dilithium, we can utilize three types of permutation dependency. When using Full-Dependence permutation, a single permutation can be used for each KeyGen and Sign algorithm. On the other hand, when using Independence permutation, each target function in the KeyGen and Sign algorithms that requires shuffling countermeasures should use independent permutations. Lastly, to use Group-Dependence permutation, we need to establish criteria for grouping the functions first.

The secret key of CRYSTALS-Dilithium consists of four secret values: K, s_1, s_2, and t_0. The relationships among these secret values are as follows:

$$As_1 + s_2 = t\,(= (t_1, t_0)) \tag{1}$$

and y and w are generated using the secret value K. In line 17 of Algorithm 2, y is added to the result of multiplying c and s_1. Similarly, in line 22, w is subtracted from the result of multiplying c and s_2. While the result of adding y is publicly disclosed, the result of subtracting w is not. Therefore, it is possible to recover s_1 from K, but not s_2, due to the lack of public information about the subtraction operation.

Based on the interrelationships among the four secret values, we can group the operations that need countermeasures in Algorithm 1 and Algorithm 2. We can group the relevant operations in Algorithm 1 as follows:

- Group 1:
  - Lines 2 and 13, where K (related to s_1) is used.
  - Lines 4 to 8 and 13, where s_1 is generated or used.
- Group 2:
  - Lines 4 and 13, where s_2 is generated or used.
- Group 3:
  - Lines 10, 11, and 13, where t_0 is generated or used.
- Group 4:
  - Line 9, where both s_1 and s_2 are used together. This group should be separated from the groups where s_1 or s_2 are used solely.

Similarly, grouping the operations in Algorithm 2 can be done as follows:

- Group 1:
  - Lines 0, 4, and 15 to 18, where s_1 is used.
  - Lines 0 and 5 to 11, where K (related to s_1) is used.
- Group 2:
  - Lines 0 and 4, where s_2 is used.
- Group 3:
  - Lines 0, 4, and 25 to 28, where t_0 is used.
- Group 4:
  - Lines 20 to 24, where both K and s_2 are used together.
- Group 5:
  - Lines 29 and 30, where K, s_2, and t_0 are used together.

In this case, there is an additional step before performing the operations in Algorithm 2, which is the unpacking process of the secret key, sk. Let's denote this step as a new line, line 0, which represents the process of unpacking the secret key. This step can be similar to the process described in line 13 of Algorithm 1.

By using 4 groups in the KeyGen algorithm and 5 groups in the Sign algorithm to apply Group-Dependence permutation, even if one group is analysed, it is not possible to determine the permutations of other groups based on the analysed permutation information. Therefore, the grouping criteria we defined are valid.

Table 2 illustrates the operations requiring shuffling countermeasures in the key generation algorithm of CRYSTALS-Dilithium along with their operation counts. Sampling, NTT, NTT^{-1}, multiplication, reduction, addition, rounding, and packing are the targeted operations. Table 3 presents the operations necessitating shuffling countermeasures in the signature algorithm, along with their operation counts. Unpacking, NTT, NTT^{-1}, sampling, multiplication, reduction, decompose, addition, subtraction, norm check, and MakeHint are the operations targeted. Since both algorithms involve multiplication operations with a dimension of $k \times l \times n$, we can use a maximum of 3-D permutation. However, since both k and l represent the number of polynomials,

TABLE 2. The detailed iteration count of the target operations in Algorithm 1.

Operation position	Target variable or equation	Operation count
2, 4. Sampling	K, s_1, s_2	K: 32 s_1: $l \times n$ s_2: $k \times n$
5. NTT 8. NTT^{-1}	s_1 $\hat{t}$	s_1: $l \times n/2$ $\hat{t}$: $k \times n/2$ and n
6. Multiplication	$\widehat{A}\widehat{s_1}$	$k \times l \times n$
7, 10. Reduction	$\hat{t}, t$	$k \times n$
9. Addition	$t + s_2$	$k \times n$
11. Rounding	Power2Round	$k \times n$
13. Packing	K, s_1, s_2, t_0	K: 32 s_1: $l \times n/8$ or $n/2$ s_2: $k \times n/8$ or $n/2$ t_0: $k \times n/8$ or $n/2$

TABLE 3. The detailed iteration count of the target operations in Algorithm 2.

Operation position	Target variable or equation	Operation count
0. Unpacking	K, s_1, s_2, t_0	K: 32 s_1: $l \times n/8$ or $n/2$ s_2: $k \times n/8$ or $n/2$ t_0: $k \times n/8$ or $n/2$
4, 6. NTT 9, 16, 21, 26. NTT^{-1}	s_1, s_2, t_0, y $\widehat{w}, \widehat{z}, \widehat{r_0}, \widehat{h}$	s_1, y: $l \times n/2$ s_2, t_0: $k \times n/2$ $\widehat{z}$: $l \times n/2$ and n $\widehat{w}, \widehat{r_0}, \widehat{h}$: $k \times n/2$ and n
5. Sampling	ExpandMask	$l \times n/4$ or $n/2$
7, 15, 20, 25. Multiplication	$\widehat{A}\widehat{y}, \widehat{c}\widehat{s_1}, \widehat{c}\widehat{s_2}, \widehat{c}\widehat{t_0}$	$\widehat{A}\widehat{y}$: $k \times l \times n$ $\widehat{c}\widehat{s_1}$: $l \times n$ $\widehat{c}\widehat{s_1}, \widehat{c}\widehat{t_0}$: $k \times n$
8, 10, 18, 23, 27. Reduction	$\widehat{w}, w, z, r_0, h$	$\widehat{w}, w, r_0, h$: $k \times n$ z: $l \times n$
11. Decompose	Decompose	$k \times n$
17, 29. Addition 22. Subtraction	$z + y, r_0 + r_1$ $w - r_0$	$z + y$: $l \times n$ $r_0 + r_1$: $k \times n$ $w - r_0$: $k \times n$
24, 28. Norm Check	r_0, r_1	$k \times n$
30. MakeHint	MakeHint	$k \times n$

we will consider 1-D and 3-D permutations and omit 2-D permutations.

In addition, for the transformations to and from the NTT domain in **CRYSTALS-Dilithium**, we modify the shuffling method proposed by Ravi et al. in 2020 [12]. Ravi et al. proposed three shufflings specifically for the NTT: Coarse-Full-Shuffle, Coarse-In-Group-Shuffle, and Fine-Shuffle. However, Hermelink et al. in 2023 demonstrated that both Coarse-In-Group-Shuffle and Fine-Shuffle are vulnerable to attacks when applied to **CRYSTALS-Kyber** [22]. Although the NTT used in **CRYSTALS-Dilithium** utilizes all eight layers and does not allow sparse NTT inputs as in Hermelink et al.'s attack, we still analyse that it is vulnerable to attacks with high complexity. Therefore, we adopt Coarse-Full-Shuffling for our shuffling. However, in the implementation code of Ravi et al., different permutations were used for each layer. Since using different permutations does not enhance security, for the sake of efficiency, we choose to use the same permutation for the layers within the NTT operation.

Using the three permutation dependencies (Full-Dependence, Independence, and Group-Dependence) and two permutation dimensions (1-D and 3-D), we can construct six different horizontal shufflings for **CRYSTALS-Dilithium**. These can be represented as six points on the plane formed by the permutation dependency axis and the permutation dimension axis, as shown in Figure 4.

4) VERTICAL SHUFFLING

To apply vertical shuffling, the functions or groups of functions should possess the characteristics mentioned in Section III-B. However, the KeyGen algorithm of **CRYSTALS-Dilithium** does not have a relevant part for vertical shuffling. In the Sign algorithm, there are two possible parts that satisfy the criteria. The first one is the NTT transformation operations of s_1, s_2, and t_0, which occur in line 4 and are all independent and identical functions. These operations can be shuffled vertically. The second part, which consists of groups from line 15 to 19, line 20 to 24, and line 25 to 28, requires some modifications to satisfy the criteria for vertical shuffling. However, the group from line 25 to 28 does not include any addition or subtraction operations, which are required for vertical shuffling. Therefore, it cannot be directly shuffled vertically. Considering the difference in the number of polynomials between s_1, s_2, and t_0 in different security levels of **CRYSTALS-Dilithium**, it is necessary to address the distinguishability issue that arises from the different polynomial counts. To address this issue, Algorithm 3 is presented, which modifies the sign algorithm to apply vertical shuffling while considering the varying number of polynomials in s_1, s_2, and t_0. This modified algorithm ensures that the shuffling is performed in a way that mitigates the distinguishability through side-channel trace observation.

In Algorithm 2, line 4 is divided into lines 4 and 5 in Algorithm 3. In line 4, only the NTT transformation functions for l polynomials of s_1, s_2, and t_0 are performed, shuffling their execution order. Then, in line 5, the NTT transformations for the remaining $k-l$ polynomials of s_2 and t_0 are performed, shuffling their execution order as well.

In Algorithm 2, lines 15 to 28 are replaced by lines 15 to 39 in Algorithm 3. Similar to the preceding NTT transformation function, lines 20 to 29 are first executed for l polynomials, while lines 30 to 39 handle the remaining $k-l$ polynomials. There are two differences from Algorithm 2. Firstly, in line 19, the norm check function undergoes horizontal shuffling. In Algorithm 2, horizontal shuffling was applied to the norm check function for r_0 and r_1, as they handle secret values, but not for z, which is a public value. However, to ensure indistinguishability in the side-channel trace, Algorithm 3 applies horizontal shuffling to the norm

---

Algorithm 3 **CRYSTALS-Dilithium**: Sign With Vertical Shuffling

Input: Message m,
Secret key $sk = (\rho, K, tr, s_1, s_2, t_0)$
Output: Signature $\sigma = (z, c, h)$

⋮

4: $\kappa := 0, \widehat{s_1} := \mathrm{NTT}(s_1)$,
$\widehat{s_2}[0:l-1] := \mathrm{NTT}(s_2[0:l-1])$,
$\widehat{t_0}[0:l-1] := \mathrm{NTT}(t_0[0:l-1])$
5: $\widehat{s_2}[l:k-1] := \mathrm{NTT}(s_2[l:k-1]), \widehat{t_0}[l:k-1] := \mathrm{NTT}(t_0[l:k-1])$

⋮

15: $\widehat{z} := \widehat{c}\widehat{s_1}$
16: $z := \mathrm{NTT}^{-1}(\widehat{z})$
17: $z = z + y$
18: $z = z \bmod^{+} Q$
19: If $\|z\|_\infty \geq \gamma_1 - \beta$, then goto line 5
20: $\widehat{r_0}[0:l-1] := \widehat{c}\widehat{s_2}[0:l-1]$
21: $r_0[0:l-1] = \mathrm{NTT}^{-1}(\widehat{r_0}[0:l-1])$
22: $r_0[0:l-1] = w[0:l-1] - r_0[0:l-1]$
23: $r_0[0:l-1] = r_0[0:l-1] \bmod^{+} Q$
24: If $\|r_0[0:l-1]\|_\infty \geq \gamma_2 - \beta$, then goto line 5
25: $\widehat{h}[0:l-1] := \widehat{c}\widehat{t_0}[0:l-1]$
26: $h[0:l-1] = \mathrm{NTT}^{-1}\left(\widehat{h}[0:l-1]\right)$
27: $r_1[0:l-1] := h[0:l-1] \bmod^{+} Q$
28: $d[0:l-1] = d[0:l-1] - h[0:l-1]$
29: If $\|r_1[0:l-1]\|_\infty \geq \gamma_2$, then goto line 5
30: $\widehat{r_0}[l:k-1] := \widehat{c}\widehat{s_2}[l:k-1]$
31: $r_0[l:k-1] = \mathrm{NTT}^{-1}(\widehat{r_0}[l:k-1])$
32: $r_0[l:k-1] = w[l:k-1] - r_0[l:k-1]$
33: $r_0[l:k-1] = r_0[l:k-1] \bmod^{+} Q$
34: If $\|r_0[l:k-1]\|_\infty \geq \gamma_2 - \beta$, then goto line 5
35: $\widehat{h}[l:k-1] := \widehat{c}\widehat{t_0}[l:k-1]$
36: $h[l:k-1] = \mathrm{NTT}^{-1}\left(\widehat{h}[l:k-1]\right)$
37: $r_1[l:k-1] := h[l:k-1] \bmod^{+} Q$
38: $d[l:k-1] = d[l:k-1] - h[l:k-1]$
39: If $\|r_1[l:k-1]\|_\infty \geq \gamma_2$, then goto line 5

⋮

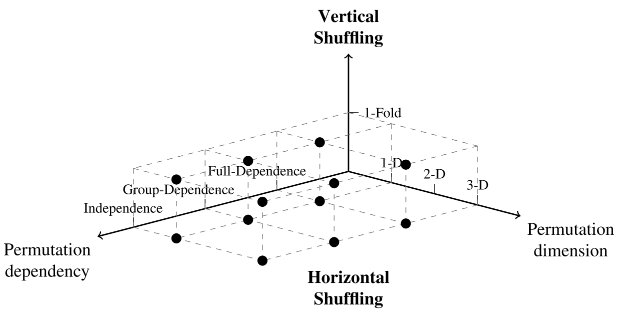


FIGURE 4. Proposed shufflings on shuffling space.

check function for z as well. Secondly, dummy subtraction operations are added in lines 28 and 38 of Algorithm 3. These dummy operations are included to maintain the same structure as other function groups. In the modified algorithm, vertical shuffling is applied by shuffling three groups: lines 15 to 19, lines 20 to 24, and lines 25 to 29. Then, two groups are shuffled: lines 30 to 34 and lines 35 to 39, to further apply vertical shuffling.

By applying the vertical shuffling described in this section to the horizontal shuffling explained in Section III-B, we can add six points with 1-Fold vertical shuffling to the shuffling space illustrated in Figure 4. In this way, we can explore a total of 12 shuffling methods that can be applied to CRYSTALS-Dilithium. These shuffling methods provide different combinations of horizontal and vertical shuffling to enhance the security of the algorithm.

B. IMPLEMENTATION RESULTS

We utilized the PQClean implementation [21] of CRYSTALS-Dilithium as our baseline code in this study. The code was compiled using clang-1403.0.22.14.1 on a MacBook Air equipped with an Apple M1 chip. We employed the `-O3` compilation option for optimization purposes. To generate permutations, we leveraged the Knuth-Yates algorithm, as implemented in the code provided by Ravi et al. [12]. For performance measurement, we utilized neon-ntt[1] and calculated the average number of clock cycles over one thousand executions.

Table 4, 5, and 6 showcase the performance measurement outcomes for the three security levels, encompassing all twelve combinations where both techniques were employed. The 3D permutation, along with group dependence, was utilized, and the inclusion of vertical shuffling was denoted as "3-D Group-Dependence + 1-Fold". It should be noted that vertical shuffling is solely applicable to the sign algorithm; thus, the performance was solely measured for the sign algorithm. Upon applying 1-D Full-Dependence to security level 5, the results demonstrated the lowest overhead, with a 12.4% increase in KeyGen and a 17.18% increase in Sign. When exclusively employing horizontal shuffling with a 1-dimensional permutation, the overhead

[1] https://github.com/neon-ntt/neon-ntt

TABLE 4. Performance of shuffling on CRYSTALS-Dilithium security level 2.

Countermeasures	Cycles			
	KeyGen	Overhead (%)	Sign	Overhead (%)
Dilithium2				
Unprotected	180,452	-	757,681	-
1-D Full-Dependence	229,897	27.40	920,289	21.46
1-D Group-Dependence	288,529	59.89	1,039,516	37.20
1-D Independence	371,895	106.09	2,137,121	182.06
3-D Full-Dependence	716,368	296.99	1,430,945	88.86
3-D Group-Dependence	937,303	419.42	1,747,666	130.66
3-D Independence	1,070,294	493.12	6,865,549	806.13
1-D Full-Dependence + 1-Fold	-	-	968,267	27.79
1-D Group-Dependence + 1-Fold	-	-	1,108,420	46.29
1-D Independence + 1-Fold	-	-	2,323,799	206.70
3-D Full-Dependence + 1-Fold	-	-	1,504,179	98.52
3-D Group-Dependence + 1-Fold	-	-	1,801,120	137.71
3-D Independence + 1-Fold	-	-	7,271,065	859.65

TABLE 5. Performance of shuffling on CRYSTALS-Dilithium security level 3.

Countermeasures	Cycles			
	KeyGen	Overhead (%)	Sign	Overhead (%)
Dilithium3				
Unprotected	349,661	-	1,193,358	-
1-D Full-Dependence	404,957	15.81	1,452,356	21.70
1-D Group-Dependence	468,847	34.09	1,619,647	35.72
1-D Independence	560,292	60.24	2,950,470	147.24
3-D Full-Dependence	1,191,013	240.62	2,255,246	88.98
3-D Group-Dependence	1,577,035	351.02	2,749,554	130.40
3-D Independence	1,827,311	422.60	11,879,878	895.50
1-D Full-Dependence + 1-Fold	-	-	1,592,589	33.45
1-D Group-Dependence + 1-Fold	-	-	1,620,553	35.80
1-D Independence + 1-Fold	-	-	3,449,518	189.06
3-D Full-Dependence + 1-Fold	-	-	2,396,105	100.79
3-D Group-Dependence + 1-Fold	-	-	3,246,429	172.04
3-D Independence + 1-Fold	-	-	12,555,134	952.08

remained below 60%. While the presented outcomes may appear relatively substantial, it is crucial to consider that the implemented countermeasures cover not only the targeted operations of the attacks proposed for CRYSTALS-Dilithium, but also operations involving secret values that have yet to be attacked. Naturally, as the permutation dimension increases, the permutation becomes more independent, and vertical shuffling is introduced, the overhead proportionally escalates.

Indeed, the most secure countermeasures are 3-D Independence for KeyGen and 3-D Independence + 1-Fold for Sign. However, the overhead of most secure methods are very high. Hence, making a proper choice of shuffling

TABLE 6. Performance of shuffling on CRYSTALS-Dilithium security level 5.

Countermeasures	Cycles			
	KeyGen	Overhead (%)	Sign	Overhead (%)
Dilithium5				
Unprotected	522,549	-	1,533,335	-
1-D Full-Dependence	587,349	12.40	1,796,812	17.18
1-D Group-Dependence	646,325	23.69	1,923,810	25.47
1-D Independence	731,291	39.95	2,908,841	89.71
3-D Full-Dependence	1,978,983	278.72	3,278,560	113.82
3-D Group-Dependence	2,411,002	361.39	3,834,584	150.08
3-D Independence	2,698,693	416.44	14,308,616	833.17
1-D Full-Dependence + 1-Fold	-	-	1,925,679	25.59
1-D Group-Dependence + 1-Fold	-	-	1,943,160	26.73
1-D Independence + 1-Fold	-	-	3,316,648	116.30
3-D Full-Dependence + 1-Fold	-	-	3,320,547	116.56
3-D Group-Dependence + 1-Fold	-	-	4,408,809	187.53
3-D Independence + 1-Fold	-	-	15,088,532	884.03

countermeasure is essential for a countermeasure designer to strike a balance between security and performance considering specific environment of the target cryptographic device and assumptions about potential attackers.

V. CONCLUSION

In this study, we have introduced a comprehensive and systematic framework for shuffling techniques. Unlike previous research that focused on proposing specific shuffling methods for particular algorithms or operations, our framework provides a structured approach for designers to identify suitable shuffling characteristics for cryptographic algorithms, considering their unique properties. This enables designers to make well-informed decisions when selecting appropriate countermeasures.

Moreover, we applied our framework to CRYSTALS-Dilithium, which is one of the PQC signature algorithms selected as a finalist in NIST's PQC standardization process. We developed and implemented twelve shuffling countermeasures and evaluated their performance. Importantly, we applied these countermeasures to functions that have not been previously targeted by attacks. The results demonstrated a minimal overhead of 12.4%, indicating the feasibility and effectiveness of our proposed shuffling techniques.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

[2] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. (2016). *Report on Post-Quantum Cryptography*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

[3] G. Alagic et al. (2022). *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf

[4] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS–kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 353–367.

[5] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Silder, and D. Stehlé. (2020). *Crystals-Dilithium: Algorithm Specifications and Supporting Documentation*. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[6] P. C. Kocher, "Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 1996, pp. 104–113.

[7] G. Alagic et al. (2020). *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8309/final

[8] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, vol. 31. Springer, 2008.

[9] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Proc. 23rd Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, Santa Barbara, CA, USA, Springer, Aug. 2003, pp. 463–481.

[10] C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," in *Proc. 4th Int. Conf. Appl. Cryptogr. Netw. Secur. (ACNS)*. Singapore: Springer, Jun. 2006, pp. 239–252.

[11] S. Lee, S. M. Cho, H. Kim, and S. Hong, "A combined single trace attack on global shuffling long integer multiplication and its novel countermeasure," *IEEE Access*, vol. 8, pp. 5244–5255, 2020.

[12] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable SCA countermeasures against single trace attacks for the NTT: A performance evaluation study over Kyber and Dilithium on the arm cortex-M4," in *Proc. 10th Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng. (SPACE)*, Kolkata, India, Springer, Dec. 2020, pp. 123–146.

[13] 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. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 583–590.

[14] K. Ngo, E. Dubrova, and T. Johansson, "Breaking masked and shuffled CCA secure saber KEM by power analysis," in *Proc. 5th Workshop Attacks Solutions Hardw. Secur.*, Nov. 2021, pp. 51–61.

[15] A. Berzati, A. C. Viera, M. Chartouni, S. Madec, D. Vergnaud, and D. Vigilant, "A practical template attack on CRYSTALS-Dilithium," Cryptol. ePrint Arch., Tech. Rep., 2023.

[16] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert, "Shuffling against side-channel attacks: A comprehensive study with cautionary note," in *Proc. 18th Int. Conf. Theory Appl. Cryptol. Inf. Secur. Adv. Cryptol. (ASIACRYPT)*, Beijing, China, Springer, Dec. 2012, pp. 740–757.

[17] N. Veshchikov, S. F. Medeiros, and L. Lerman, "Variety of scalable shuffling countermeasures against side channel attacks," *J. Cyber Secur. Mobility*, vol. 5, no. 3, pp. 195–232, 2017.

[18] S. Tillich, C. Herbst, and S. Mangard, "Protecting AES software implementations on 32-bit processors against power analysis," in *Proc. 5th Int. Conf. Appl. Cryptogr. Netw. Secur. (ACNS)*, Zhuhai, China, Springer, Jun. 2007, pp. 141–157.

[19] M. Rivain, E. Prouff, and J. Doget, "Higher-order masking and shuffling for software implementations of block ciphers," in *Proc. 11th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, Lausanne, Switzerland, Springer, Sep. 2009, pp. 171–188.

[20] B.-C. Nguyen and C.-K. Pham, "A combined blinding-shuffling online template attacks countermeasure based on randomized domain Montgomery multiplication," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2022, pp. 1–6.

[21] M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P-Workshops)*, Genoa, Italy. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2022, pp. 19–30. [Online]. Available: https://eprint.iacr.org/2022/337

[22] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, "Adapting belief propagation to counter shuffling of NTTs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 60–88, Nov. 2022.

JONGHYEOK LEE received the B.S. degree in mathematics from Kookmin University, Seoul, South Korea, in 2017, where he is currently pursuing the Ph.D. degree in financial information security. He specializes in information security. His research interests include deep learning-based side-channel analysis, fault injection attack, and countermeasures against side-channel analysis.

JAESEUNG HAN received the M.S. degree in financial information security from Kookmin University, Seoul, Republic of Korea, in 2022, where he is currently pursuing the Ph.D. degree in financial information security. His research interests include side-channel analysis, symmetric key cryptography, post-quantum cryptography, and countermeasures against side-channel analysis.

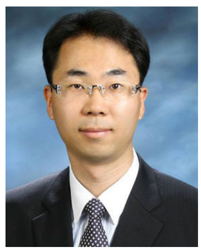
SANGYUB LEE received the B.S. degree in telecommunications and the M.E. and Ph.D. degrees in information security from Korea University, Seoul, Republic of Korea, in 2011, 2015, and 2020, respectively. He is currently a Senior Engineer with the Security Algorithm Team, Samsung SDS Inc., Seoul. His research interests include side-channel attacks, public-key cryptosystems, and embedded system security.

JIHOON KWON received the B.S. degree in mathematics and the Ph.D. degree in information security from Korea University, Seoul, South Korea, in 2010 and 2018, respectively. He is currently a Senior Engineer with the Security Algorithm Team, Samsung SDS Inc., Seoul. His research interests include cryptography and information security and their efficient implementation.

KEON-HEE CHOI received the B.S. degree in information security, cryptology, and mathematics from Kookmin University, Seoul, South Korea, in 2023, where he is currently pursuing the M.S. degree in financial information security. His research interests include side-channel analysis and post-quantum cryptography.

JAE-WON HUH received the B.S. degree in information security, cryptology, and mathematics from Kookmin University, Seoul, South Korea, in 2022, where he is currently pursuing the M.S. degree in financial information security. His research interests include side-channel analysis, symmetric key cryptography, fault analysis, and post-quantum cryptography.

JIHOON CHO received the M.Math. degree in cryptography from the University of Waterloo, and the Ph.D. degree in information security from the Royal Holloway University of London. He was a Security Architect of mobile devices with LG Electronics, South Korea. He is currently the Director of the Security Research Team, Samsung SDS Inc., South Korea.



DONG-GUK HAN received the B.S. and M.S. degrees in mathematics and the Ph.D. degree in engineering in information security from Korea University, Seoul, Republic of Korea, in 1999, 2002, and 2005, respectively. He was a Postdoctoral Researcher with Future University Hakodate, Hokkaido, Japan. After finishing his Ph.D., he was an Exchange Student with the Department of Computer Science and Communication Engineering, Kyushu University, Japan, from April 2004 to March 2005. From 2006 to 2009, he was a Senior Researcher with the Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea. He is currently a Professor with the Department of Information Security, Cryptology, and Mathematics, Kookmin University, Seoul. He is a member of KIISC, IEEK, and IACR.