Fully Parallel, One-Cycle Random Shuffling for Efficient Countermeasure in Post-Quantum Cryptography

Jong-Yeon Park, Dongsoo Lee, Seonggyeom Kim, Wonil Lee, Bo Gyeong Kang, and Kouichi Sakurai *Member, IEEE*

Abstract—Hiding countermeasures are the most widely utilized techniques for thwarting side-channel attacks, and their significance has been further emphasized with the advent of Post Quantum Cryptography (PQC) algorithms, owing to the extensive use of vector operations. Commonly, the Fisher-Yates algorithm is adopted in hiding countermeasures with permuted operation for its security and efficiency in implementation, yet the inherently sequential nature of the algorithm imposes limitations on hardware acceleration. In this work, we propose a novel method named Addition Round Rotation (ARR), which can introduce a time-area trade-off with block-based permutation. Our findings indicate that this approach can achieve a permutation complexity level commensurate with or exceeding 2^{128} in a single clock cycle while maintaining substantial resistance against second-order analysis. To substantiate the security of our proposed method, we introduce a new validation technique – *Identity Verification*. This technique allows theoretical validation of the proposed algorithm's security and is consistent with the experimental results. Finally, we introduce an actual hardware design and provide the implementation results on Application-Specific Integrated Circuit (ASIC). The measured performance demonstrates that our proposal fully supports the practical applicability.

Index Terms—Permutation, Shuffling, Side-channel attack, Post Quantum Cryptography

1 Introduction

1.1 Background and Motivation

In the post-quantum era, all the currently used public key encryption algorithms, such as RSA [1], Elliptic Curve Cryptography (ECC), and Digital Signature Algorithm (DSA) [2], could be potentially broken by Shor's algorithm [3]. Because these public key algorithms are primitives for digital signatures, message and user authentication in most security applications, it is necessary to standardize alternatives that are resistant to Shor's algorithm.

The algorithms selected for Post Quantum Cryptography (PQC) are based on lattice, hash, and code-based problems [4], [5], [6], [7]. Main operations for these algorithms include a significant amount of vector operations that allow for parallel processing. This implies the parallelizable structures of most PQC algorithms are more advantage for applying countermeasures against side channel attack compared to RSA and ECC, where the integer operations are the basic structures.

Hiding countermeasures are basically defined as techniques to reduce correlation between side channel leakages and intermediate values processed by the executed device [8]. The technical coverage of hiding countermeasure has many branches; from cell level to algorithm level. Most widely used technique is the operation shuffling that can be applied by both software and hardware in algorithm level. Our research focuses on the operation shuffling for hiding countermeasure.

Shuffling has been applied as supplementary countermeasures to masking countermeasures in symmetric key encryption like Advanced Encryption Standard (AES [9], [10]). The reason why shuffling is hardly used as the main countermeasure is the small number of parallel operations; for example, there are only 16 Sboxes that can perform a randomly permuted operation in AES. However, the polynomial dimension of PQC algorithms can range from 32 to several thousands. In this case, solely used shuffling countermeasure can provide enough resistance against side channel attack even in really powerful assumption of attackers. Also, while masking countermeasures can theoretically be neutralized by single trace attacks, shuffling is considered an effective countermeasure against them.

Although shuffling for PQC algorithms seems to be easily adopted as the countermeasures, there are two issues to be resolved – the efficient way of mixing a sequence and the easy security evaluation. In this regard, we address both issues and present a novel technique allowing an optimal trade-off between the security level and performance.

1.2 Related Works and Challenging Issues

The most common method of mixing a sequence of a certain size is to shuffle the sequence randomly using a random permutation. The most widely used algorithm for the random permutation generation is the Fisher-Yates (F-Y shuffling) algorithm, also commonly known as Knuth shuffling [11], [12]. The greatest advantage of F-Y shuffling is that it provides fully factorial complexity with simple operations.

However, F-Y shuffling has the following disadvantages. Firstly, a permutation must be sequentially generated. This makes hardware parallel acceleration impossible. Secondly, presequence must be stored in advance, so it necessarily occupies a certain amount of memory. Lastly, due to the algorithmic feature that N elements must be mixed with independent rule, it needs a relatively large amount of random numbers. Moreover, this high randomness cost for a large amount of indices can be a significant loss when implementing it in hardware. Therefore, a more efficient shuffling method for a hardware implementation is required. Z. Chen et al. [13] provide a case where high-efficiency shuffling is applied using addition and 1-bit swap for an unit on number theoretic transformation (NTT) against side channel attack. Such shuffling expresses a most $2^{128} \times 128$ distinct sequences on 256 indices. However, it is not secure from the perspective of side-channel analysis, because each index is completely interconnected. This approach has limitations in the security against certain attacks such as Soft Analytical Side-Channel Attacks (SASCA) [14], [15], [16], [17]. According to

- *Jong-Yeon Park, Dongsoo Lee, Seonggyeom Kim, Wonil Lee, Bo-Gyeong Kang are engineers in Samsung electronics System LSI Business E-mail: jonyeon.park@samsung.com*
- *Kouichi Sakurai is in Kyushu University*

H.Julius et al. [17], all layers in operations in NTT need to be shuffled for the optimal security.

Small domain encryption can also be understood as a permutation. The most extensively studied topic in this area is Format Preserving Encryption (FPE), and the research focuses on encryption schemes where input size is operated flexibly. Prior to the standardization of FPE with FFX [18], various studies were conducted, including encryption operating with a card shuffling, block ciphers [19], [20], fixed ciphers utilizing table lookups [21], and methods that flexibly use input and output by using modes of operations of block ciphers [18]. However, research for the provable security of FPE focuses on distinguishability from a random permutation for a specific size of query. This approach provides security from the perspective of encryption and decryption rather than the entropy from the entire permutation complexity perspective. It is different from the direction of our study, which must discuss the entire entropy or complexity of the permutation. Furthermore, for very small permutations for shuffling where the total number of indices is less than or equal to 2^{12}, to the best of our knowledge, all suggested algorithms of FPE require too many operations.

As an independent research topic, the study of generating random permutations in parallel has also been approached in various ways [22], [23], [24], [25], [26]. This field of research exhibits two main characteristics. Firstly, it combines parallelism with sequentiality, failing to maximally leverage hardware efficiency based on complete parallelism. Secondly, all these approaches aim solely to generate entirely uniform permutations (with the complexity of $N!$), meaning that if the algorithm is stopped midway, there will be some indices that are not shuffled at all (or complexity is not proven). Therefore, it has the limitation using the trade-off between complexity and efficiency. As a result, it is inefficient for use in security contexts and excessive random numbers required.

Closest to our work is a permutation based parallel algorithm operated on GPU (graphics processing unit) with bijection with Feistel network [27], [28]. The suggested algorithm has a similar approach to our fully parallel random shuffling. This research shows the block bijection can generate uniform random permutations which pass their own statistical test criteria; however, it does not accurately represent complexity. Additionally, because their research cannot compute the actual complexity of permutations, and passing their test does not mean it has the attack complexity of a factorial.

Alternatively, shuffling can be implemented using the Routing Network such as Benes network [29], [30] with a little variation. The advantages of the network-based approach are its ability to execute parallel operations at high speeds and its capability to generate entire permutations. However, a significant drawback is the excessive random numbers required. Also, for the decoding implementation without "Encoding and Decoding" and parallel operation, the size of hardware must be even larger than general Benes Network. Also, it needs more research parallel operation without "Encoding and Decoding" step [31]. We will discuss this topic in section 6.

Hence, we need to research an efficient scheme that operates within the required complexity bounds in a cryptography industry and provides the security level against exhaustive permutation investigation and countermeasures against side channel attack.

1.3 Our Contributions with New Shuffling Techniques

We propose an efficient and secure permutation generator, namely ARR, and a new security evaluation method, namely *identity verification* to ensure that the proposed approach is secure. According to our evaluation method, ARR can achieve the sufficient attack complexity, and can be efficiently implemented in hardware. ARR iterates Add-Rotate-XOR (ARX) block cipher-like round functions [32]. Thus, it can be considered to be a Markov process with a transition matrix. The diffusion speed can be analyzed through mixing time, spectral gap [33], etc. However, constructing the transition matrix to analyze Markov process is an extremely difficult task since the sample size is too large. Therefore, we propose a method called *Identity Verification*, which allows for measuring the entropy at a specific round of the permutation. Also, *Identity Verification* can be generalized to the estimation of entropy for every permutation with symmetric structures.

The contributions of this paper are as follows:

- We introduce a shuffling method ARR and theoretically analyze it. Also, we introduce important theoretical properties.
- We propose a novel technique termed *Identity Verification* to compute attack complexity. Through actual experiments implementing this technique, we can calculate the theoretical complexity of the proposed algorithm. Furthermore, we prove the efficacy of this approach through rigorous mathematical proof.
- We propose a hardware design on ASIC for a single clock cycle operation ARR and analyze its area and performance. Moreover, we provide a variation of ARR with practical security against side channel attack.

2 PREREQUISITE

2.1 Side Channel Attack and Hiding countermeasure

The basic implementation for the hiding countermeasure with shuffling is as follows:

1) Identify operations that can be shuffled. For instance, AES Sbox operations can be performed in arbitrary order, making them eligible for shuffling.
2) Generate a non-repetitive random sequence as large as the number of the operations to be shuffled.
3) Perform the operations in the according order to the generated sequence.

The security of hidden operations in Step 3 totally depends on the complexity of shuffled sequence generated in Step 2. This paper verifies whether the proposed permutation function has 'full complexity or total entropy' of permutation under a single trace attack which can have $N!$ as the number of cases. For multiple trace attacks, it checks the quality of the permutation through the entropy of 'First-order' and 'Second-order' attacks, it is also known as first-order and second-order permutation [34]. First-order attacks are the initial considerations in the context of shuffling countermeasures against adversaries with single location target, and second-order attacks serve as a primary indicator to verify the shuffling in obfuscating the relationships among each index against adversaries with multiple location targets.

2.2 Fisher-Yates Shuffling

Algorithm 1 illustrates the most widely used method to generate a shuffled sequence, F-Y shuffling. The most prevalent reason for its usage is its ability to create permutations following a uniform distribution, with a complexity of $N!$. This

Algorithm 1 Fisher-Yates Shuffling

Input: A ordered Set
$X = \{0, 1, ..., N-1\} = \{X_0, X_1, ..., X_{N-1}\}$.
Output: A ordered Set
$Y = \{Y_0, Y_1, ..., Y_{N-1}\}$ where $Y_i \in \mathbb{Z}_N$.
1: **for** $j = N$ to 2 **do**
2: Choose a random integer k such that $0 \leq k < j$.
3: Exchange the values of X_{k-1} and X_{j-1}
4: $Y_{j-1} \leftarrow X_{j-1}$
5: **end for**
6: **return** Y

technique involves storing the sequence in advance and conducting a step-by-step transposition (c.f., line 3 of Algorithm 1). The outcome of each step then determines the operations in the next phase. Consequently, this method cannot be represented by a fixed permutation function for each index, and it's not possible to generate the entire sequence in parallel all at once.

Once the shuffled sequence has been produced, operations $\mathsf{OP}(\cdot)$ can proceed in the generated order. In order words, this changes the order of N operations as

$$\{\mathsf{OP}(0), \mathsf{OP}(1), \ldots, \mathsf{OP}(N-1)\} \mapsto \{\mathsf{OP}(Y_0), \mathsf{OP}(Y_1), \ldots, \mathsf{OP}(Y_{(N-1)})\},$$

where $(Y_0, Y_1, \ldots, Y_{(N-1)})$ is a random sequence.

2.3 Notations and Background of Permutation

For convenience, we first define notations for the following sections.

A permutation is a function that is one-to-one correspondence. Moreover, we limit the domain as integer space $\mathbb{Z}_N$. Thus, by the definition of permutation, the range of output is also $\mathbb{Z}_N$.

$$\mathsf{Perm} : K \times \mathbb{Z}_N \to \mathbb{Z}_N$$

refers to a family of permutations transforming integers. Therefore, Perm has the property that, for each $k \in K$, the projection $\mathsf{Perm}(k, \cdot)$ is a permutation on $\mathbb{Z}_N$. The realizations of Perm will appear as ϕ, δ, or ARR. The set K represents a key set which determines the diversity of the permutation with given method Perm. Thus, $\mathsf{Perm}(K, x) = y$ denotes a set of *pseudo random permutations* that are determined by a key set K including random numbers $\{k_i\}$. If we just say $\mathsf{Perm}(x)$ without K, then it assumes that K is a randomly chosen. In this paper the term *random permutation* means a pseudo random permutation. 'I' is the identity permutation, $I(x) = x$ for all x, $\mathsf{Perm}(K, \cdot)$ can be the identity permutation probably.

Additionally, we will employ the cyclic notation for permutations, which is one of the ways to represent permutations in Algebra. The notation is expressed as (A, B, C), signifying that through the given function, A is transformed to B, B to C, and C back to A. In the same way, a *transposition* (A, B) means, A is transformed B, and B back to A.

We will denote the probability of an event E occurring as $\Pr(E)$. Also, in this paper, all references to 'log' denote the logarithm base 2. This is used for a computation for entropy, we say $H(X)$ with a random variable X [35]. An entropy is computed as,

$$H(X) = -\sum_{x \in X} \Pr(x) \log(\Pr(x)) \quad (1)$$

$H(X)$ is the computational time complexity for the guessing secret information with log scale, as $2^{H(X)}$.

In this paper, we intend to use the following symbols. A function representation $\boxplus_k(t)$ signifies modular addition $(k+t) \mod N$ with $k, t \in \mathbb{Z}_N$, where the number used for the modulus will be defined at the point of usage of this symbol. However, if we implement the algorithm in a real system, N is defined as 2^n, we do not need to operate modular but only remove the final carry bits.

$(\ll (t))$ or $(\gg (t))$ represent a 1-bit left rotation and right rotation (not shift) of a value t, respectively. Rotation refers to the process of moving the overflowed bit to the opposite side after the shift. Above operations, $\boxplus, (\ll), (\gg)$ can be used as binary operation symbol, $A \boxplus B = (A + B) \mod N$, $(A \ll 1)$ is the same meaning to $(\ll (A))$.

3 Block-based Permutation : Addition-Rotation Round (ARR)

3.1 Structure description

In this section, we explain the proposed method; Addition-Rotation Round (ARR) . Our design principle for good structures is based on the following five conditions:

1) **Efficiency**. To maximize operational speed, processing should be possible in fully parallel, and each index should be able to operate independently.
2) **Simple and Low Cost**. In order to optimize the implementation area, it should be combined using the simplest operations.
3) **Divergence**. It can generate all permutations, and is well diffused.
4) **Provability**. To determine the level of security, the complexity should be provable.
5) **Ease**. To prevent hardware or software developers from making mistakes in implementation, the design should be easy to understand.

To maximize diffusion in each round, at least two requirements need to be met, first, a variation interval dependent on the key is necessary, akin to the AddRoundKey phase in AES. Second, there should be a combination of at least two operations. If not, an internal subgroup may be formed in the permutation group, thereby reducing the number of cases. For example, with only AddRoundKey, hundreds of rounds of operations are equivalent to just one. Third, these two or more operations should neither commute nor combine. If the operations are commutative, the number of rounds can be eliminated mathematically. If they can be combined, there could be fewer cases than the size of the input key due to linearity.

In light of five conditions, we propose the ARR (Addition ($\boxplus$) Rotation ($\ll$) Round) method, which operates in the manner shown in Algorithm 2. We identify the following key features:

1) **Round Function**. To maximize area efficiency through the repetition of certain operations, we have applied a round function, a common methodology in Block Cipher.
2) **Fixed Rotation**. Efficiently designing a 'variable rotation' necessitates an additional area for each rotation. From a hardware design perspective, fixed rotations can be regarded as having no operations since they can be implemented by re-wiring. Also, no random numbers are needed for variability.

3) **Modular Addition**. It is a simple operation which makes variations depending on the key value.

There are various methods of operation depending on the keys; such as exclusive-or ($\oplus$) or modular addition ($\boxplus$). These operations are widely employed in the block cipher design [9], [32], [36]. Among various options, we have selected fixed 1-bit rotation and modular addition as our operations for the following reasons: First, these operations allow for a unified structure of operations regardless of bit size. Second, they do not obey commutativity and linearity. For example, if the operations were composed of exclusive-or and 1-bit rotation, they would have a linearity relationship, which is to say, $(A \ll 1) \oplus (B \ll 1) = (A \oplus B) \ll 1$.

Another option would be a combination of exclusive-or and modular Addition operations. In this case, while having a remarkably simple structure. Furthermore, the modular addition and exclusive-or operations do not exhibit a linearity relationship. However, the combination cannot generate full permutation, but a sub-group of group of all permutations S_n. For example, in 2 bits operation, $N = 4$ has $4! = 24$ cases for permutations, but $\oplus$ and $\boxplus$ combination can generate only 8 elements which is a sub-group of S_n which is known as dihedral group D_8 [37].

Algorithm 2 Addition-Rotation Round ($\mathsf{ARR_R}$) - Generalized form with Round R

Input: An integer $x \in \mathbb{Z}_N$, Key set $K = \{k_0, k_1, ..., k_R\}$, $k_i \in \mathbb{Z}_N$, and the even number Round R
Output: $y = \mathsf{ARR_R}(K, x)$.
1: $t \leftarrow x$
2: **for** $i = 0$ to $(R/2) - 1$ **do**
3: $\quad t \leftarrow \boxplus_{k_i}(t)$
4: $\quad t \leftarrow (\lll (t))$
5: **end for**
6: **for** $i = (R/2)$ to $R - 1$ **do**
7: $\quad t \leftarrow \boxplus_{k_i}(t)$
8: $\quad t \leftarrow (\ggg (t))$
9: **end for**
10: $y \leftarrow \boxplus_{k_R}(t)$
11: **return** y

Algorithm 3 Generating a full sequence with ARR (Can be Executed in Parallel)

Input: Key set $K = \{k_0, k_1, ..., k_R\}$, Round R.
Output: A random sequence $Y = \{y_0, y_1, y_2, ..., y_{N-1)}\}$, $y_i \neq y_j$ where $i \neq j$, denote $\mathsf{ARR_R}(K, \cdot)$
1: **for** $i = 0$ to $N - 1$ **do**
2: $\quad y_i \leftarrow \mathsf{ARR_R}(K, i)$
3: **end for**
4: **return** Y

Finally, we choose modular addition ($\boxplus$) and ($\lll$) and ($\ggg$) combination and we call it Addition-Rotation Round (ARR). The operation of the entire algorithm is described in Algorithm 2. The fundamental structure is characterized by the simple repetition of modular addition and a fixed rotation, with a distinctive mid-round reversal of the shift direction (Line 6), as well as an additional final addition (Line 10). This structure is intentionally designed for its symmetric properties. The theoretical advantages of such symmetry for validation purposes will be discussed in the following section. The generation function for creating a permutation over the entire input can be formulated as shown in Algorithm 3, allowing for the creation of one element at a time. Of course, because each operation is independent, it is also possible to execute them in parallel. Ultimately, we propose the adoption of a structure that only carries out the operation of a half round, see Algorithm 4. To avoid confusion, we will refer to the ARR operation for the full R round as $\mathsf{ARR_R}$, and the ARR operating for only half rounds as $\mathsf{ARR_{HR}}$.

Algorithm 4 Addition-Rotation Round ($\mathsf{ARR_{HR}}$) - Generalized form with the Half-Round HR

Input: An integer $x \in \mathbb{Z}_N$, Key set $K = \{k_0, k_1, ..., k_{R/2}\}$, the even number Round R
Output: $y = \mathsf{ARR_{HR}}(K, x)$
1: **for** $i = 0$ to $(R/2) - 1$ **do**
2: $\quad t \leftarrow \boxplus_{k_i}(t)$
3: $\quad t \leftarrow (\lll (t))$
4: **end for**
5: $y \leftarrow \boxplus_{k_{R/2}}(t)$
6: **return** y

3.2 Stationary distribution and Divergence of ARR

ARR should possess diversity, which is an essential characteristic required for security.

1) **Divergence**. It generates all permutations. (**Theorem 1**)
2) **Uniformness**. When it is repeated, it arrives at a *stationary distribution* as the uniform distribution. This necessitates the fulfillment of the following properties which described in **Proposition 1**, according to ergodic theory in Markov chain [38].
3) **Fast Diffusion**. It must rapidly diffuse in proportion to the size of the input random numbers. (Section 4).

To prove Proposition 1, the following three conditions are met. Firstly, the operation should be capable of generating all possible permutations with repeated rounds. This capability is identified as the property of *irreducibility*. Secondly, the operation should exhibit 'returning to itself', namely *symmetric*, see Definition 1. Lastly, the operation should possess an *aperiodic* property, implying that transitions to each state should not be periodic. This is a naturally occurring property when every statement can return to its initial state. In our structure, aperiodicity is naturally seen to be true, since ARR becomes an identity function with zero keys.

Definition 1. A random function ϕ is said to be *symmetric* if $\Pr(\phi(x) = y) = \Pr(\phi(y) = x)$

Proposition 1. The stationary distribution of ARR performed repeatedly converges uniform distribution.

Proof. The random permutation ARR is symmetric, aperiodic and irreducible. Then, the stationary distribution of the Markov chain converges uniform distribution. □

The symmetry can also be observed as a structural characteristic of ARR, as only the shift direction is reversed after the half-round, see Algorithm 4. We will address a variety of properties related to symmetry in a Section 3.3.

Now, we only need to show the irreducibility. Before demonstrating the capability of the ARR to generate all permutations, we would like to mention an widely accepted statement (Fact 1) concerning permutations and transpositions [37].

Fact 1. Every permutation can be transformed into a combination of transpositions. A transposition is a permutation that exchanges two indices.

In other words, every permutation is a composition of transpositions. If all transpositions can be generated, it is equivalent to being able to generate all permutations. We will prove that ARR can generate all transpositions.

Lemma 1. For any given random permutation ϕ that generates a transposition $(x, x+1)$ for a particular $x \in \mathbb{Z}_N$. Let $\delta = \boxplus_k \circ \phi \circ \boxplus k^{-1}$ for arbitrary random number $k \in \mathbb{Z}_N$, then δ can generate every transposition of the form $(y, y+1)$ for any $y \in \mathbb{Z}_N$.

Proof. Let's consider a transposition $\tau = (x, x+1)$ such that $\tau(x) = x+1$ and $\tau(x+1) = x$. Now, we observe that $(\boxplus_k \circ \tau \circ \boxplus_{k^{-1}})(\boxplus_k(x)) = (\boxplus_k \circ \tau(x)) = \boxplus_k(x+1)$ and $(\boxplus_k \circ \tau \circ \boxplus_{k^{-1}})(\boxplus_k(x+1)) = \boxplus_k(x)$. Also, $(\boxplus_k \circ \tau \circ \boxplus_{k^{-1}})(\omega) = \omega$ such that ω is not $\boxplus_k(x+1)$ and $\boxplus_k(x)$.

Consequently, $(\boxplus_k \circ \tau \circ \boxplus_{k^{-1}})$ can be represented in the cyclic form of the transposition $(\boxplus_k(x), \boxplus_k(x+1))$. By the definition of $\boxplus_k$, $(\boxplus_k(x), \boxplus_k(x+1)) = ((x+k), (x+1+k))$. Let $x+k$ be y. Since k can be any values, $(\boxplus_k \circ \tau \circ \boxplus_{k^{-1}}) = (\boxplus_k(x), \boxplus_k(x+1))$ is able to be any transposition $(y, y+1)$.

□

Lemma 2. Let n be a bit size of input and $R \geq 2n$, $\mathsf{ARR}_\mathsf{R}(K, \cdot)$ generates the transposition $(0, 1)$ for any input bits.

Proof. First, let's define define a permutation Ω. It sets the all keys are zero up to $(n-1)th$-rounds, and then add (-1) in n-th round. Finally, output value is the result of after $(+2)$ which is last key. In other words, Ω is written as $(n-1)$ times left rotation, addition (-1), one time left rotation, and $(+2)$ with modular reduction by 2^n. We show that Ω derive the transposition $(0, 1)$ with the key set-up.

$\Omega(0)$ is (-1) after n rounds. After the final addition, it is (1). Thus, $\Omega(0) = 1$. Also, $\Omega(1)$ is (2^{n-1}) after $(n-1)$ rounds. At the n-th round addition, the value is $(2^{n-1} - 1)$. After the last left rotation of $(2^{n-1} - 1)$ is $(2^n - 2)$. Finally, the last addition make the value $2^n - 2 + 2 = 2^n = 0 \bmod 2^n$. Thus $\Omega(1) = 0$.

Now, we need to $\Omega(x) = x$ for all $x \in \mathbb{Z}_{2^n}$, where $x \neq 0$ and $x \neq 1$. Assume that x is even. Left $(n-1)$ rotation is the same to one right rotation. Thus, it is $(x/2)$ after left $(n-1)$ rotation. At the last round addition (-1), it becomes $(x/2 - 1)$. After last left rotation and it becomes $(x-2)$, because MSB of $(x/2-1)$ is zero. Finally, $\Omega(x) = x - 2 + 2 = x$ when x is even.
Assume that x is odd, it is $2^{n-1} + (x-1)/2$ after left $(n-1)$ rotation. At the last round addition (-1), it becomes $2^{n-1} + (x-1)/2 - 1$. After last left rotation, it becomes $2((x-1)/2 + 2^{n-1} - 1) \bmod (2^n - 1)$. (Remark, left rotation can be described modular reduction by $(2^n - 1)$).

$$\begin{aligned} & 2((x-1)/2 + 2^{n-1} - 1) \mod (2^n - 1) \\ &= ((x-1) + 2^n - 2) \mod (2^n - 1) \\ &= \{(x-1) \mod (2^n - 1) + 2^n \mod (2^n - 1) + \\ & \quad (-2) \mod (2^n - 1)\} \mod (2^n - 1) \end{aligned}$$

We will check the three parts,
$(x-1) \mod (2^n - 1) = (x-1)$ since $(x-1) < 2^n - 1$.
$(2^n) \mod (2^n - 1)$ is 1. $(-2) \mod (2^n - 1) = 2^n - 3$. Therefore,

$$\begin{aligned} & ((x-1) + 1 + 2^n - 3) \mod (2^n - 1) \\ &= (x + 2^n - 3) \mod (2^n - 1) \\ &= (x + 1 - 3) \mod (2^n - 1) = (x-2) \mod (2^n - 1) \\ &= (x-2) \end{aligned}$$

After the last addition $(+2)$, then $\Omega(x) = x - 2 + 2 = x$. In conclusion, $\Omega(0) = 1$, $\Omega(1) = 0$, and $\Omega(x) = x$ for all x where $x \neq 0$ and $x \neq 1$. Thus, Ω is the transposition $(0, 1)$. Now, we can say ARR_R can generate the transposition $(0, 1)$ for all input bit length n. □

Lemma 3. Given a permutation ϕ represented by $\mathsf{ARR}_R(K_0, \cdot)$ for some $K_0 \in \mathbb{Z}_N$, there exists a $K_1 \in \mathbb{Z}_N$ such that $\boxplus_k \circ \phi \circ \boxplus_k^{-1}$ is represented by $\mathsf{ARR}_R(K_1, \cdot)$ for all $k \in \mathbb{Z}_N$, provided that R is sufficiently large.

Proof. By the definition of $\mathsf{ARR}_\mathsf{R}(K, \cdot)$, it generate $\boxplus_k$ only before and after ϕ with $\log(N)$ round. Thus, $\boxplus_k \circ \delta \circ \boxplus_{-k}$ is generated. Since $\boxplus_{-k}$ is equivalent to $\boxplus_k^{-1}$, the proof is done. □

Theorem 1. The random permutation $\mathsf{ARR}_\mathsf{R}(K, \cdot)$ represents all possible permutations with sufficiently large R.

Proof. By Lemma 1, 2, and 3, $\mathsf{ARR}_\mathsf{R}(K, \cdot)$ generates all transpositions $(y, y+1)$, where $y \in \mathbb{Z}_N$. By composition of transpositions $(y, y+1) \circ (y+1, y+2) \circ (y, y+1) = (y, y+2)$ with the transposition $(y, y+1)$ and $(y+1, y+2)$ which are generated by $\mathsf{ARR}_\mathsf{R}(K, \cdot)$. Now, we have $(y, y+2)$. We can generate another transposition with the same manner, $(y, y+2) \circ (y+2, y+3) \circ (y, y+2) = (y, y+3)$. In this way, $\mathsf{ARR}_\mathsf{R}(K, \cdot)$ generate $(y, y+k)$, where $k \in \mathbb{Z}_N$. Thus, $\mathsf{ARR}_\mathsf{R}(K, \cdot)$ represents any transposition. By **Fact 1**, the proof is done. □

By Theorem 1, we can confirm that ARR satisfies irreducibility. Thus, it fulfills all three properties mentioned in Proposition 1 - irreducibility, aperiodicity, and symmetry. In other words, given that the proposed structure carries a uniform distribution in the long run and can possess all permutations, we recognize it as a viable structure for security purposes, assuming the cost of increasing rounds is the only consideration.

3.3 Identity verification for Complexity evaluation

To evaluate the speed at which ARR disperses to encompass all possible permutations, we propose a novel approach accompanied by experimental validation. The number of permutations significantly surpasses intuitive estimates. For instance, there are $40,320$ permutations $(8!)$ for just 3 bits (8 input indices), and the number of permutations $(16!)$ exceeds 2^{44} for 4 bits (16 input indices).

In light of these considerations, conducting a conventional statistical analysis for situations with more than 4 bits is impractical. For a comprehensive statistical analysis, for example χ^2-test, a minimum required size of storage is approximately $83 \times c$ TB (assuming each case requires 4 bytes, although it is generally expected to be more). Moreover, a minimum of $16!$ test iterations implies a requirement of $80 \times c$ days (assuming a 3GHz CPU, 1000 clock cycles per test loop, with mandatory file input/output which is also expected to consume more time, c is the constant for statistical power). Thus, we propose an efficient approach for statistical analysis.

According to our method,

- No storage space is required at all.
- The analysis time is much shorter.

Furthermore, for cases of lower complexity with small rounds, testing can be terminated quickly. For this, we define a new concept called *Identity Verification* which estimate *Identity Probability* (Definition 2) and will prove mathematically why this analysis is feasible. Due to the symmetric structure, there must exist a case where it becomes an identity function, as described in Section 3.2.

Definition 2. *Identity Probability* for a given random permutation, denoted by $\Pr(\mathsf{Perm}(X, \cdot) = I)$ with random variable X, is

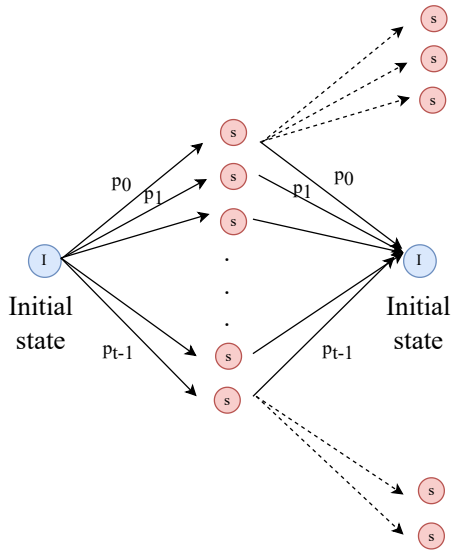


Fig. 1: Probabilities and Statements of a symmetric permutation

defined as the probability that the permutation of the identity results in the identity itself. *Identity Verification* is a statistical test to estimate *Identity Probability*.

Our ultimate finding is that if there exists a symmetric random permutation, the lower bound of computational complexity can be determined using *Identity Probability* when only half rounds are performed. In conclusion, we will calculate the actual *Identity Probability* through experimentation, and combining the results of this experiment with theory. Finally, we will propose the final algorithm necessary to achieve the desired complexity. We will start from Proposition 2 that is based on the concept of Figure 1.

Proposition 2. For two random permutations ϕ and δ satisfying $\Pr(\phi(x) = y) = \Pr(\delta(y) = x)$, and given all potential output permutations $\phi = \{s_0, s_1, s_2, ..., s_{t-1}\}$, we define $\Pr(\phi = s_i)$ as p_i. Then, the identity probability of $((\delta \circ \phi) = I)$ is $\sum_{i=0}^{t-1} p_i^2$, denote, $\Pr((\delta \circ \phi) = I) = \sum_{i=0}^{t-1} p_i^2$, where $\circ$ represents function composition.

Proof. Given $\phi = \{s_0, s_1, s_2, ..., s_{t-1}\}$, we can express the probability $\Pr((\delta \circ \phi) = I)$ as follows:

$$\Pr((\delta \circ \phi) = I) = \sum_{i=0}^{t-1} \Pr(\delta = {s_i}^{-1} | \phi = s_i)$$

Since $\phi = s_i$ and $\delta = {s_i}^{-1}$ are independent, and $\Pr(\phi(x) = y) = \Pr(\delta(y) = x) = \Pr(\delta^{-1}(x) = y)$,
$\Pr(\delta = {s_i}^{-1} | \phi = s_i) = \Pr(\delta^{-1} = s_i | \phi = s_i) = \Pr(\phi = s_i | \phi = s_i)$ simplifies to p_i^2. Therefore, we obtain

$$\Pr((\delta \circ \phi) = I) = \sum_{i=0}^{t-1} p_i^2 \tag{2}$$

□

Corollary 1. All possible output permutations of $\mathsf{ARR_{HR}}$ as $\{s_0, s_1, s_2, ..., s_{t-1}\}$. If $\Pr(\mathsf{ARR_{HR}} = s_i) = p_i$, then $\Pr(\mathsf{ARR_R} = I) = \sum_{i=0}^{t-1} p_i^2$.

Proof. Let's consider the two parts of the ARR_R operation: $\mathsf{ARR_{HR}}$ and the operations outside of $\mathsf{ARR_{HR}}$ in ARR_R, which we will call $\mathsf{ARR_{ER}}$. Hence, we can describe the operation flow of $\mathsf{ARR_R}$ as $\mathsf{ARR_{HR}} \rightarrow \mathsf{ARR_{ER}}$. The overlapping addition is cancelled out by the initial step of $\mathsf{ARR_{ER}}$ since two redundant additions are merged to a single addition. Therefore, $\mathsf{ARR_R} = \mathsf{ARR_{ER}} \circ \mathsf{ARR_{HR}}$. Given the definitions of $\mathsf{ARR_{HR}}$ and $\mathsf{ARR_R}$, it follows that $\mathsf{ARR_{ER}}$ is the reverse of $\mathsf{ARR_{HR}}$. Consequently, $\Pr(\mathsf{ARR_{ER}}(x) = y)$ equals $\Pr(\mathsf{ARR_{HR}}(y) = x)$, which is proven by Proposition 2. □

The following Corollary 1 naturally derives from the proposition 2, and it can be seen that our proposed structure, ARR also satisfies Equation 2. Now, we will prove Theorem 2, which enable us to calculate the complexity of every permutation with symmetric structures by *Identity Verification*.

Theorem 2. Let ϕ and δ be two random permutations such that $\Pr(\phi(x) = y) = \Pr(\delta(y) = x)$, and let $q = \Pr((\delta \circ \phi) = I)$. Then, the entropy $H(X)$ of the random variable X that follows the distribution $\phi(x)$ satisfies the inequality

$$H(X) \geq -\log(q) \tag{3}$$

Proof. Let's denote the key space (the number of random keys) of ϕ as T, and let the set of all possible states with T random keys, $\phi = \{s_0, s_1, ..., s_{t-1}\}$, where $t \leq T$. Suppose that each state s_i occurs x_i times, which yields:

$$\Pr(\phi = s_i) = \frac{x_i}{T} \tag{4}$$

and

$$\sum_{i=0}^{t-1} x_i = T \tag{5}$$

By Proposition 2, it follows that the sum of squared probabilities equals the probability of identity, i.e., $(x_0/T)^2 + (x_1/T)^2 + (x_2/T)^2 + ... + (x_{t-1}/T)^2 = \Pr(\phi = I) = q$. This can be rewritten as:

$$\sum_{i=0}^{t-1} x_i^2 = T^2 q \tag{6}$$

Let's calculate the entropy, $H(X) = -\sum \Pr(x) \log(\Pr(x))$ with Equation 4 and 5. It can be computed as follows:

$$\begin{aligned} H(x) &= -\sum_{i=0}^{t-1} \frac{x_i}{T} \log((x_i/T)) \\ &= -1/T \sum_{i=0}^{t-1} x_i (\log(x_i) - \log(T)) \\ &= -\frac{1}{T} \sum_{i=0}^{t-1} x_i (\log(x_i)) + \frac{1}{T} \sum_{i=0}^{t-1} x_i \log(T) \\ &= -\frac{1}{T} \sum_{i=0}^{t-1} x_i (\log(x_i)) + \frac{\log(T)}{T} \times T \\ &= \log(T) - \sum_{i=0}^{t-1} \frac{x_i}{T} (\log(x_i)) \end{aligned}$$

The logarithm function is concave, and by Jensen's inequality [39] and Equation 6, we have:

$$\sum_{i=0}^{t-1} \frac{x_i}{T} (\log(x_i)) \leq \log \sum_{i=0}^{t-1} \frac{x_i^2}{T} = \log(Tq)$$

Thus, we get:

$$-\log(Tq) \leq -\sum_{i=0}^{t-1} \frac{x_i}{T} (\log(x_i))$$

Finally, we can derive the lower bound for the entropy:

$$H(x) = \log(T) - \sum_{i=0}^{t-1} \frac{x_i}{T}(\log(x_i)) \geq \log(T) - \log(Tq)$$
$$= -\log(q)$$

□

Theorem 2 can be naturally applied to our structure, ARR, as shown in Corollary 2, and we can see from Corollary 3 that it can be generally applied to all symmetric permutations. We will use Theorem 2 to compute the complexity with experiments in Section 4.1.

Corollary 2. Suppose ARR_R is a random permutation such that $\Pr(\mathsf{ARR}_\mathsf{R} = I) = q$ and let X be a random variable following the distribution of ARR_HR. Then, the entropy of ARR_HR is at least $-\log(q)$.

Proof. This is consequently proven by Theorem 2 and Corollary 1. □

Corollary 3. Assume ϕ is a symmetric permutation as defined in Definition 1, and $\Pr((\phi \circ \phi) = I) = q$. Then, the entropy of ϕ is at least $-\log(q)$.

Proof. This is consequently proven by Theorem 2 and Definition 1. □

This means that the theory we propose is not only useful for testing the initial complexity of our structure but also of all types of permutations with a symmetric form, indicating potential for wide-ranging applications in various fields.

Remark 1. By the Corollary 2, once the lower bound of the attack complexity verified up to 'Half Round' is determined, it is sufficient for us to apply operations only up to ARR_HR to generate permutations satisfying this complexity.

Also, we can get the final fact Remark 1. From now on, we will use practically only the Half-round algorithm. Full round algorithm is only used for verification of the Half-Round algorithm.

3.4 Small Units Aggregation (SUA)

Although Identity Verification is much better tool than standard statistical tests, approximately 40 bits is practical limit with lab-level PC. Consequently, we chose a practical strategy of verifying our structure up to 40 bits and then extending it, rather than generalizing our structure.

Algorithm 5 displays examples of *Small Unit Aggregation* implemented in four parts, denote 4-SUA. The reason for dividing into four is as follows.

Firstly, it is possible to increase the entropy diffusion rate with more key size. Secondly, Identity Verification allows us to test up to around 40 bits. For example, if a complexity of more than 32 bits is guaranteed with the four independently permuted blocks, the complexity of exhaustive search is 128 bits (32×4). This is equivalent to the complexity of brute force searching for master key of AES 128 [9], which is considered cryptographically secure of 2023.

The mathematical explanation for the complexity of one part being raised to the fourth power when divided into four parts will be omitted here; we show intuitive description with Figure 2. It shows 4 bits example (16 balls), it is separated by two Most Significant Bits (MSB) and two Least Significant Bits (LSB). By the selection of MSB, LSB extra bits out of MSB 2 bits

Algorithm 5 Addition-Rotation Round (ARR) - Generalized form with Half-Round HR with four Small Units Aggregation (4-SUA)

Input: An integer $x \in \mathbb{Z}_N$, $s = \log(N)$
Key set $K_0 = \{k_{0,0}, k_{0,1}, ..., k_{0,R/2}\}$,
Key set $K_1 = \{k_{1,0}, k_{1,1}, ..., k_{1,R/2}\}$,
Key set $K_2 = \{k_{2,0}, k_{2,1}, ..., k_{2,R/2}\}$,
Key set $K_3 = \{k_{3,0}, k_{3,1}, ..., k_{3,R/2}\}$,
every $k_{i,j}$ is $(s-2)$ bits size.
the even number Round R,
Output: $y = \mathsf{ARR}_\mathsf{HR}(K, x)$ with 4-SUA
1: $C \leftarrow$ Most Significant 2 bits of x
2: $t \leftarrow x$
3: **for** $i = 0$ to $(R/2) - 1$ **do**
4: $t \leftarrow \boxplus_{k_{C,i}}(t)$ /* (mod 2^{s-2}) */
5: $t \leftarrow (\lll (t))$ /*(rotation LSB $(s-2)$ bits)*/
6: **end for**
7: $y \leftarrow \boxplus_{k_{C,R/2}}(t)$
8: **return** y

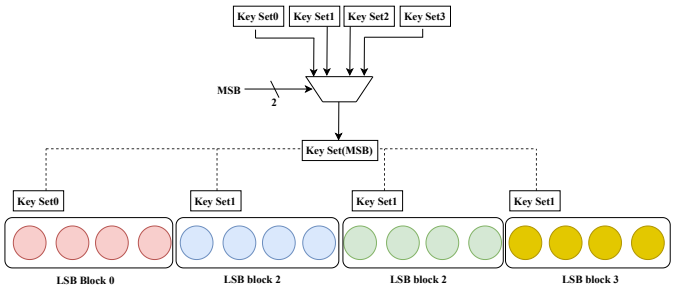


Fig. 2: Permutation with Four Small Unit Aggregation (4-SUA)

is only computed by ARR iteration with the selected key set. (While we may not compute in this way with 4-bit examples, it serves merely for illustrative purposes). In reality, we would begin from 6 bits, subdividing into four groups based on the top 2 bits MSB. We were to generate a 10-bit permutation with partitioning 8 bits into four groups. The variable C (Line 1) in Algorithm 5 leads to division into four separate parts of the same size, and it can be seen that the shuffling occurs independently in these four separated spaces.

Therefore, we have managed to generate a permutation, applicable to any bit size, that ensures at least the total complexity of AES 128, and this can be accomplished at a relatively low cost.

3.5 Second-Order Diffusion (SOD) for Security

Critical aspects of the permutation used in side-channel analysis is the complexity of the first-order and second-order multiple trace attacks. In the case of first-order attack, it can be easily solved by the initial addition for the entire size. Therefore, applying only initial addition alone guarantees sufficient security against first-order Differential Power Analysis [40]. However, after utilizing 4-SUA against second-order attack, second-order complexity is reduced to a quarter, see the 4-separated balls in Figure 2.

Algorithm 6 shows ARR with Second-Order attack Diffusion (SOD) applied. This algorithm represents our final proposed form, incorporating first-order diffusion (Line 1, 2) and enhancing second-order resistance. By varying the approach with the

Algorithm 6 Addition-Rotation Round (ARR) - Generalized form with Half-Round HR with 4-SUA for Second-order diffusion (SOD)

Input: An integer $x \in \mathbb{Z}_N$, $s = \log(N)$,
Key set $K_0 = \{k_{0,0}, k_{0,1}, ..., k_{0,R/2}\}$,
Key set $K_1 = \{k_{1,0}, k_{1,1}, ..., k_{1,R/2}\}$,
Key set $K_2 = \{k_{2,0}, k_{2,1}, ..., k_{2,R/2}\}$,
Key set $K_3 = \{k_{3,0}, k_{3,1}, ..., k_{3,R/2}\}$,
every $k_{i,j}$ is $(s-2)$ bits size,
Initial Key $\lambda \in \mathbb{Z}_N$.
the even number Round R,
Output: $y = \mathsf{ARR}_{\mathsf{HR}}(K, x)$ with 4-SUA and SOD

$t \leftarrow \boxplus_{\lambda}(t)$, ($\bmod N$)
$r \leftarrow$ a random number s bits
$t \leftarrow$ Left rotation r bits of t
$C \leftarrow$ Most Significant 2 bits of t
for $i = 0$ to $(R/2) - 1$ **do**
  $t \leftarrow \boxplus_{k_{C,i}}(t)$ /* ($\bmod 2^{s-2}$) */
  $t \leftarrow (\ll (t))$ /*(rotation LSB $(s-2)$ bits)*/
end for
$t \leftarrow \boxplus_{k_{C,R/2}}(t)$
$y \leftarrow$ Right rotation r bits of t.
return y

4-SUA method each time, SOD complicates the relationships between each permuted index. This can be effective in countering attacks that employ techniques such as SASCA [17].

4 Security

4.1 Total Permutation Complexity

In this section, we discuss the attack complexity of the permutation. We used *Identity Verification* presented in Section 3.3 for full rounds Algorithm 2, to compute the lower bound of entropy for the half round. Complexity calculations of our final structure are based on the application of 4-SUA and SOD.

Our experimental procedure is as follows:

1) We count the number of instances where $\mathsf{ARR}_{\mathsf{R}} = I$ for each round without SUA and SOD, which is described in Algorithm 3. We stop the test when this count reaches 30 (N-Hits $= 30$) and then compute the probability, $\Pr(\mathsf{ARR}_{\mathsf{R}} = I) =$ (N-Hits)/(N-Trials) with the total number of trials (N-Trials) required to reach this number. However, if (N-Hits $< 2^{24}$) when (N-Hits $= 30$), the test continues until (N-Hits $= 2^{24}$).
2) Given that the highest attack complexity that we need to know is 2^{36}, we halt the experiment if the number of trials exceeds 2^{38} and count the number of instances where $\mathsf{ARR}_{\mathsf{R}} = I$, even if (N-Hits ≤ 30).
3) To compute the complexity, we calculate $q =$ (N-Hits)/(N-Trials), and the reciprocal of this value is the attack complexity up to half rounds $\mathsf{ARR}_{\mathsf{HR}}$ by Theorem 2, denote, $1/q$.
4) The complexity of ARR with 4-SUA and SOD is $(1/q) \times 4 + n$, where n is the bit size of input domain ($+n$ due to SOD, $\times 4$ due to 4-SUA).

We present the actual experimental values for *Identity Verification* of ARR. All the experimental results in this paper have been presented based on this data in Table 1. The random keys in our experiments are derived from AES256. The input of AES operation is previous output block. The key is generated by LFSR with time seed that is built-in rand() function from Microsoft Visual C's standard library [41]. Once the key is selected, all experiments are performed by the fixed key. Ultimately, the goal of our research is to determine the number of rounds required to achieve the targeted entropy for a given input bit size.

Proposition 3. Let ϕ and δ be a set of random permutations, $\phi = \phi^{-1} = \{\phi_0 = I, \phi_1, \phi_2, ..., \phi_{M-1}\}$ such that $\phi_i \cdot \phi_j \in \phi$, and $\delta = \{\delta_0 = I, \delta_1, \delta_2, ..., \delta_{L-1}\}$ such that $\delta_i^{-1} \circ \delta_j \neq \phi_k$ for all index i, j and $k(\neq 0)$. If the entropy of ϕ and δ is X and Y, respectively, entropy of $\delta \circ \phi$ is $X + Y$.

Proof. Assume that $\delta_i \circ \phi_j = \delta_k \circ \phi_l$ for some i, j, k, l. Then, $\delta_k^{-1} \circ \delta_i = \phi_l \circ \phi_j^{-1} = \phi_u$. By the fact $\delta_i^{-1} \circ \delta_j \neq \phi_k$ for all index i, j and $k(\neq 0)$, $\delta_k^{-1} \cdot \delta_i = \phi_0 = I$. Since $\delta_k^{-1} \cdot \delta_i = I$, $\delta_k = \delta_i$ and $\phi_j = \phi_l$. Thus, every $\delta_j \circ \phi_i$ is distinguished, and ϕ and δ is independent as sample spaces of probability distribution. The concludes the proof, as it is a known fact that the entropy of the product of two independent random variables equals the sum of their individual entropies. □

The reason for simply adding an initial key (λ in Algorithm 6) equal to the size of the input bits in the Initial Addition operation is that the addition in this stage cannot be generated through the small round unit operations in the subsequent 4-SUA. From the perspective of the entire permutation, the initial addition and small round unit can be considered to act entirely independently. The independence of these two operation blocks is straightforward and will not be further discussed in the paper. When the two blocks are independent, the overall complexity (Entropy value) is calculated by simply adding the complexities of the two, based on Proposition 3. In Proposition 3, we can regard ϕ as the initial addition and δ as small unit operations with round units. The entropy of initial addition ϕ is the same to the size of input.

Table 2 displays the number of necessary required half rounds and total random bits which are used for key set to reach attack complexities of 112-bit, 128-bit, and 144-bit for input domain sizes ranging from 6-bit to 12-bit when $\mathsf{ARR}_{\mathsf{HR}}$ is applied with 4-SUA and SOA. Surprisingly, as shown in Table 2, to ensure an attack complexity of a certain bit size reaching the desired attack complexity, requires using only a nearly similar level of random bits to the attack complexity. This serves as an indicator of how much entropy scalability our method provides while performing rounds.

4.2 Second-Order Attack Complexity

The complexity of second-order attack allows us to check the relationship between indices after shuffling. We proceeded with our experiment as follows:

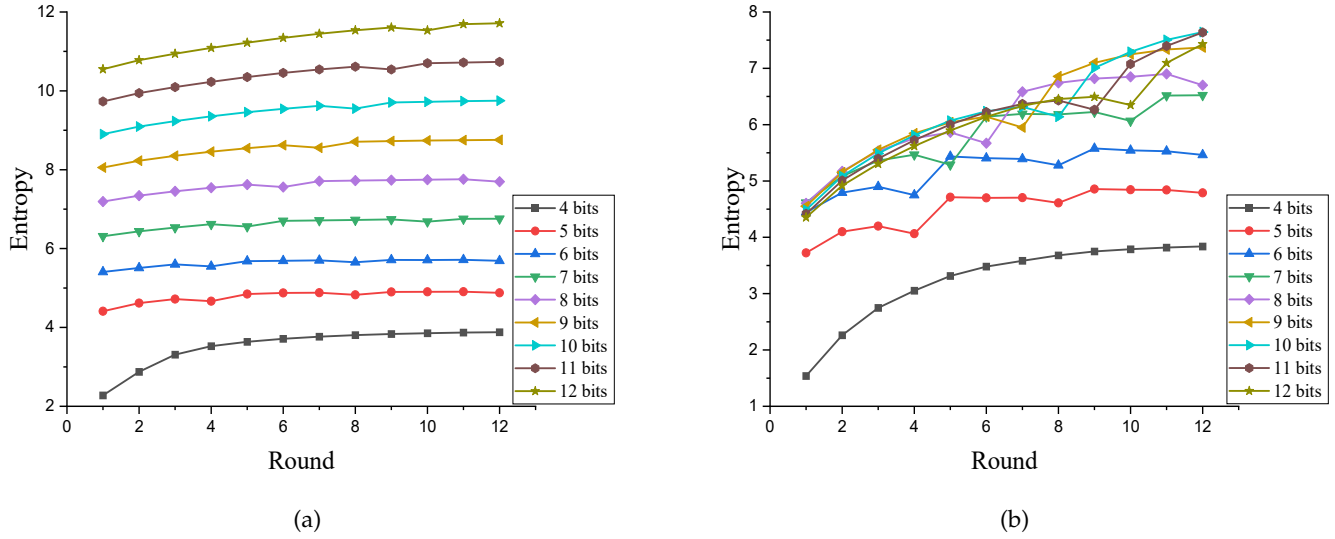
1) Classify each index according to the possible distance between each index. For instance, in the case of a 5-bit input, the distance between each index can range from 1 to 31. The number of possible distance is 31 cases.
2) Input a random number as a key and perform the $\mathsf{ARR}_{\mathsf{HR}}$ operation 10,000 times using the algorithm with 4-SUA and SOD.
3) For a fixed index (2^n cases) and the second index classified by a distance ranging from 1 to $2^n - 1$, reclassify the modified distance after $\mathsf{ARR}_{\mathsf{HR}}$ for 10,000 cases. The total number of operations for the statistical test is $2^n \times T \times 10000$.

The classification results can be summarized as follows:

1) **Worst case entropy** : This calculates the probability of the distance after the permutation. For instance, in

TABLE 1: Estimated Entropy ($H(X) = 1/q$) of Pure ARR (without SUA, SOD) for the rounds from 4 bits to 10 bits input, HR : Required half rounds, R-bits : Used key size (bits), Entropy : Entropy of ARR after half Rounds, N-Trials : The number of Trials (hex-representation), N-Hits : The number of hit ($|\mathsf{ARR}_\mathsf{R}(\mathsf{K}, \cdot) = I|$) with full round (HR×2), The symbol '-' is not experimented or not computed. ($38 \leqq$) means the entropy is larger than 38.

HR		4 bits	5 bits	6 bits	7 bits	8 bits	9 bits	10 bits
2	N-Trials (Hex)	1000000	1000000	1000000	3c24381	2ab95af8	ba98d319	7b21cdaa9
	N-Hits (Decimal)	4122	493	51	30	30	30	30
	Entropy	11.99	15.05	18.32	21.00	24.51	26.63	30.03
	R-bits	12	15	18	21	24	27	30
3	N-Trials (Hex)	1000000	19cb750	1b98e110	233475616	22a80bd055	4000000000	4000000000
	N-Hits (Decimal)	239	30	30	30	30	1	0
	Entropy	16.09	19.78	23.87	28.23	32.20	38	$38 \leqq$
	R-bits	16	20	24	28	32	36	
4	N-Trials (Hex)	1cf472b	4922cec8	96c81312f	4000000000	4000000000	-	-
	N-Hits (Decimal)	30	30	30	9	0		-
	Entropy	19.94	25.28	30.32	34.83	$38 \leqq$	-	-
	R-bits	20	25	30	35	-	-	-
5	N-Trials (Hex)	f8d5bce	5b677059e	4000000000	4000000000	-	-	-
	N-Hits (Decimal)	30	30	5	0	-	-	-
	Entropy	23.05	29.60	35.67	$38 \leqq$	-	-	-
	R-bits	24	30	36	42	48	54	60
6	N-Trials (Hex)	877908be	4000000000	-	-	-	-	-
	N-Hits (Decimal)	30	8	-	-	-	-	-
	Entropy	26.17	35	-	-	-	-	-
	R-bits	28	35	42	49	56	63	70
7	N-Trials (Hex)	2f4b74ecd	4000000000	-	-	-	-	-
	N-Hits (Decimal)	30	0	-	-	-	-	-
	Entropy	28.65	$38 \leqq$	-	-	-	-	-
	R-bits	32	40	48	56	64	72	80
8	N-Trials (Hex)	1340da717a	-	-	-	-	-	-
	N-Hits (Decimal)	30	-	-	-	-	-	-
	Entropy	31.36	-	-	-	-	-	-
	R-bits	36	45	54	63	72	81	90
9	N-Trials (Hex)	36c58eb0da	-	-	-	-	-	-
	N-Hits (Decimal)	30	-	-	-	-	-	-
	Entropy	32.86	-	-	-	-	-	-
	R-bits	40	50	60	70	80	90	100
10	N-Trials (Hex)	4000000000	-	-	-	-	-	-
	N-Hits (Decimal)	7	-	-	-	-	-	-
	Entropy	35.19	-	-	-	-	-	-
	R-bits	44	55	66	77	88	99	110
11	N-Trials (Hex)	4000000000	-	-	-	-	-	-
	N-Hits (Decimal)	0	-	-	-	-	-	-
	Entropy	$38 \leqq$	-	-	-	-	-	-
	R-bits	48	60	72	84	96	108	120



(a) (b)

Fig. 3: (a) Average entropy for all two-point relations (b) The worst case entropy for all two-point relations

TABLE 2: Required Random bits for ARR_{HR} to reach $H(X) = 2^{112}$, 2^{128} and 2^{144} where 4-SUA and SOA (Unit : Bits)

Input size(bits)	6	7	8	9	10	11	12
$H(X) = 2^{112}$	134	127	128	121	138	119	132
Required Rounds	7	5	4	3	3	2	2
$H(X) = 2^{128}$	150	147	128	149	138	155	132
Required Rounds	8	6	4	4	3	3	2
$H(X) = 2^{144}$	182	147	152	149	170	155	172
Required Rounds	10	6	5	4	4	3	3

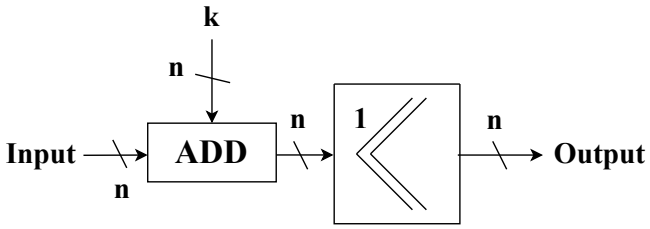


Fig. 4: Round Unit (Forward)

a 2-bit input, if the first input is 0 and the second input is 1 (distance between them is 1), and possible outputs are $\{0, 1, 2, 3\}$ each with a uniform probability of $\{0, 0.33, 0.33, 0.33\}$ (as 0 cannot occur), the entropy is $H(X) = -\sum \Pr(x) \log(\Pr(x)) = 0 + 0.528 + 0.528 + 0.528 = 1.58$. Worst case entropy calculates the entropy for each possible distance and selects the smallest value.

2) **Average entropy** : We calculate the entropy for each possible distance and compute the arithmetic mean of these entropies. This metric indirectly predicts second-order attack complexity.

Figure 3 (a) and (b) present the results for second-order. Looking at Figure 3 (a) Average entropy, the result converges towards the ideal entropy for each bit as the number of rounds increases. The maximum entropy theoretically possible is equal to the bit size. Figure 3 (b) shows the Worst case Entropy, which is generally smaller than Average entropy. However, it exhibits an overall upward trend, implying that there are no bits that stagnate and fail to increase, even with repeated rounds. Therefore, we can conclude that our permutation effectively disperses the second-order entropy.

5 Hardware Design

Our approach has been designed for the simplest computation at the smallest unit level, with the basic unit composed of addition and 1-bit shift, as already presented in the algorithm. From a hardware operation perspective, because 1-bit fixed rotation is just a re-wiring, the operation delay can be regarded as zero. Therefore, the factors affecting actual computational performance can be seen as the size of the addition unit and the number of rounds. Figure 4 shows the basic unit. The size of the addition in the basic unit is affected by the input size, and a different key is added in each round. The computation delay of a round unit is very short, allowing multiple operations within a single cycle, as shown in Figure 5.

Figure 6 shows the design working in a single clock cycle. The 4-SUA requires an extra multiplexer for key selection. SOD includes an initial addition and rotation, with a reverse rotation

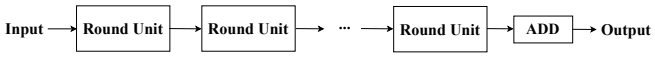


Fig. 5: Round Chaining in a single clock cycle with Round Unit

TABLE 3: Maximum number of round units operated in one clock cycle with SOD in ASIC 1GHz, 4nm Process

Input (bits)	Maximum Number of Rounds in one clock cycle
4	19
5	18
6	17
7	16
8	15
9	14
10	14
11	13
12	13

at the end, as shown in Algorithm 6. The design enables the integration of the maximum possible number of round units within a single clock cycle. Table 3 summarizes the maximum number of round units that can be included in a single clock cycle.

Adopting a different approach, instead of maximizing the number of round units within a single clock cycle, we configure the number of units to align with the required security strength. This strategy allows for an increase in operational frequency. Table 4 reports the anticipated frequencies for this configuration. The round counts used in Table 4 are based on the data reported in Table 2.

5.1 Brief summary of the result

In summary, Table 5 offers a comprehensive overview of our design result for hardware designers to effectively implement the proposed methods. To fully leverage our results, the following aspects need to be holistically evaluated: the size of the input domain, the required size of the random numbers, area of sequential and combinational logic constraints of the product under development, the performance of the required permutation, the type of pseudo-random number generator available, the complexity of attacks in a fully permutation and second-order assumption.

According to our research findings, permutations with the required attack complexity can be generated within one clock cycle. This can be remarkably efficiently utilized in hardware and software co-design scenarios [42]. When a permutation is needed during a computation, the proposed hardware architecture does not require deterministic input. It solely takes random numbers as input, which means software developers do not

TABLE 4: Maximum Operation Frequency with given complexity and SOD for a single clock cycle operation in ASIC, 4nm Process (Unit : GHz)

Input (bits)	112 bits	128 bits	144 bits
6	2.11	1.90	1.58
7	2.57	2.25	2.25
8	2.83	2.83	2.43
9	3.20	2.67	2.67
10	3.00	3.00	2.50
11	3.75	3.00	3.00
12	3.75	3.75	3.00

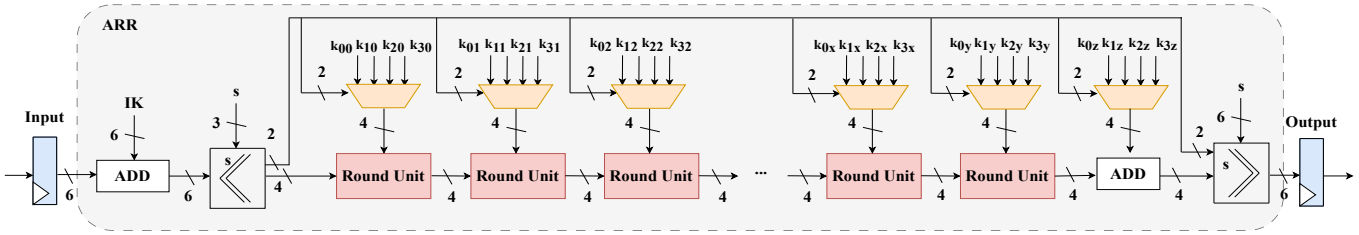


Fig. 6: Round unit Chaining 6 bits model with four 4 bits SUA (4-SUA) and SOD with Round unit - a single clock cycle operation, key indices determined by the number of rounds

TABLE 5: Brief summary of the result (ASIC, 4nm Process)
(Unit : AREA(μm^2))

	112 bits complexity						128 bits complexity						144 bits complexity					
Domain size (bits)	AREA comb.	AREA seq.	AREA Total	RNG bits	Req. Round	2nd order Avg. Entropy	AREA comb.	AREA seq.	AREA Total	RNG bits	Req. Round	2nd order Avg. Entropy	AREA comb.	AREA seq.	AREA Total	RNG bits	Req. Round	2nd order Avg. Entropy
6	24.46	41.00	65.46	134	7	5.70	26.91	45.49	72.40	150	8	5.65	31.78	54.48	86.26	182	10	5.71
7	23.88	39.59	63.47	127	5	6.56	26.89	45.21	72.10	147	6	6.70	26.89	45.21	72.10	147	6	6.70
8	24.99	24.99	49.98	128	4	7.54	24.99	24.99	49.98	128	4	7.54	28.58	47.17	75.75	152	5	7.62
9	25.63	39.03	64.66	121	3	8.35	29.83	29.83	59.66	149	4	8.46	29.83	29.83	59.66	149	4	8.46
10	28.84	44.37	73.20	138	3	9.23	28.84	44.37	73.20	138	3	9.23	33.65	33.65	67.31	170	4	9.35
11	26.72	39.59	66.31	119	2	9.94	26.72	39.59	66.31	119	2	9.94	32.19	49.70	81.90	155	3	10.10
12	30.24	43.80	74.04	132	2	10.78	30.24	43.80	74.04	132	2	10.78	36.33	55.04	91.37	172	3	10.94

need to provide additional input. It allows highly efficient generation of permutation implementations.

Additionally, we can consider reducing the number of rounds executed per clock cycle. The algorithm presented in this paper is based on performing all operations within one clock cycle, depending on the required attack complexity. However, the operations can be modified to function across 2 or 3 clock cycles by dividing the number of rounds in one clock cycle. The results in Table 5 allow for an approximate prediction of the area required for this modified implementation. This offers an additional advantage of being able to manipulate the implementation based on the resources.

6 Discussion and Limitation

6.1 4-bit, 5-bit complexity of ARR

The 4-bit complexity refers to the shuffling of the AES S-box, and it is likely to be one of the most widely used application sizes. In this case, the 4-bit ARR structure from Algorithm 4 can be directly employed, without the need to apply SOD and SUA. For the 5-bit scenario, the application of 2-SUA with 4 bits ensures 72-bit complexity after 11 rounds that is computed in one clock cycle. Since a security strength of over 76 bits is relatively small compared to 32! (which equals 117 bits) complexity, it is a limitation to our structure.

6.2 Towards Higher Complexity

To achieve greater security than the 144-bit complexity, we can use more keys and break down the data into smaller chunks. As mentioned in Section 3.4, 4-SUA splits a 6-bit section into four parts of 4 bits each, or a 7-bit section into four parts of 5 bits each. We can refine this approach: for instance, divide 7 bits into eight parts of 4 bits, or 8 bits into eight parts of 5 bits. By applying this strategy to even larger bit sizes (8-SUA), we can double the security complexity up to 288 bits.

Our approach is flexible and can be scaled up, allowing us to enhance the system by using larger key sizes. When compared to 4-SUA, 8-SUA the required space for the hardware's combination logic will be about the same, but the space for sequential logic will double due to the key space with a single clock cycles. To improve the complexity of ARR effectively, we only need to consider using more keys and the additional space needed for the larger flip-flops.

6.3 Comparison to a Structure with Feistel-based bijection

Our approach is similar to the Feistel-based structure by Mitchell et al. [27], but their structure is more complicated than ARR. This does not align with our design principle introduced in Section 3, but we will compare the results in area and performance considering the structural characteristics. The *Philox* base structure, using constant M and splitting inputs into left and right $n/2$ bits, is defined as:

$$L' = F_k(L) \oplus R$$
$$(R', b') = G(B_k(L), b).$$

where

$$F_k(R) = (M \times R)/2^W \oplus k$$
$$B_k(L) = (M \times L) \mod 2^W$$

Actually, the constant M exceeds 32 bits in size, but comparing it with a 32-bits M, the difference in performance and area with ARR is so significant that such a comparison is not meaningful. Therefore, for our comparison, we assume M to be $n/2$-bit size (ranging 3 bits to 6 bits) for our comparison. However, since we adjusted the algorithm to be smaller than

TABLE 6: Comparison of size of random bits between ARR and Benes Network

Domain size(bits)	6	7	8	9	10	11	12
	Addition Rotation Round (ARR) up to Total Entropy 2^{128}						
Applied Random bits	150	147	128	149	138	155	132
Second-Order Entropy	5.65	6.70	7.54	8.45	9.23	9.95	10.77
	Reduced Benes Network up to Total Entropy 2^{128}						
Applied Random bits	128	128	128	256	512	1024	2048
Operated Depth	4	2	1	1	1	1	1
Second-Order Entropy	3.96	1.98	0.99	0.99	0.99	0.99	0.99
	Full depth Benes Network						
Applied Random bits	352	832	1920	4352	9728	21504	47104
Operated Depth	11	13	15	17	19	21	23
Second-Order Entropy	5.976	6.98	7.99	8.91	9.74	10.70	11.66

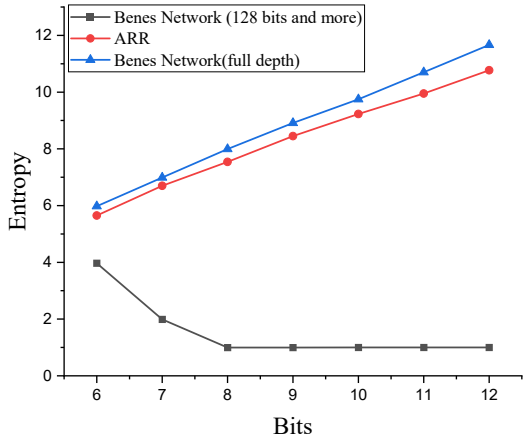


Fig. 7: Average second-order entropy of ARR, Reduced Benes Network, and Full-depth Benes network with random bits shown in Table 6

the M proposed by them, it will not satisfy their diffusion statistical test results.

Operationally, the distinction boils down to an n-bit adder in ARR versus an $n/2$-bit multiplier in *Philox*. For ASIC implementation, ARR is more area-efficient, about 1.5 to 2 times smaller, and operates 3 to 5 times faster than *Philox*. Although this varies depending on the hardware design method and synthesis results, it is very clear that our results are superior in terms of area and performance, due to the general operational characteristics of multiplication and addition. Moreover, as Table 2 shows, our method directly translates the size of input random bits into entropy, making it more effective than *Philox*'s diffusion method in reducing the number of rounds for the target complexity.

6.4 Comparison to Routing Network

Known as one of the most efficient hardware shufflers, Routing Network, e.g., the Benes Network [29], [30] has the advantage of generating a full permutation using only flip-flops, a multiplexer, and basic wiring. It facilitates the parallel execution of the entire index. However, this approach requires a large amount of random bits. If the input size is larger than 5 bits, the size of input randomness needs to be excessively large. Therefore, the method is premised on the need for a large quantity of random bits, which significantly impacts overall performance.

To use this approach with high efficiency, the amount of random bits must be reduced. A possible approach is using partial depth operation. Despite reducing depth, the complexity of exhaustive search is still sufficiently high. For example, in the case of Z. Chen et al. [13], the Benes Network is used at only one depth, and the permutation has 128-bit complexity. However, this method reveals limitations of swap-based-method since interrelationships exist between indices.

Table 6 displays three results of second-order average entropy; the Benes Network using the minimum randomness at the level of 128 bits of entropy, the full depth Benes Network, and ARR with input size 6-bit to 12-bit. The contents of the table are summarized in Figure 7. Although the reduced-depth Benes Network actually uses much more random bits than ARR, it is found that second-order entropy is much lower than that of ARR. The noteworthy result is that ARR shows entropy results comparable to the Full depth Benes Network. However, in the case of a maximum of 12 bits, the full-depth Benes Network requires 47,104 bits. On the other hand, 132 bits are sufficient for a remarkable result in ARR.

6.5 Limitations and Future works

The limitation of our research is below,

- **Area Requirement for a single clock cycle operation:** The ARR approach requires a substantial area for generating one clock cycle, necessitating flexible specification choices based on individual requirements and the usage environment.
- **Identity Verification:** The proposed Identity Verification method lacks a generalized complexity formula applicable to any given round.
- **Performance Limitations of Test PCs for verification:** The performance limitations of the test PCs restrict the complexity that can be verified in the experiments. This limitation shows that theory alone is not enough; we also need statistical experiments to fully estimate the overall complexity.

In response to these limitations, our future work will focus on the following areas,

- **Research on Area-Efficient Strategies:** There is a need for research into more area-efficient strategies to overcome the substantial area requirement for clock cycle generation.
- **Implementation on GPUs:** Similar to the approach implemented by Mitchell et al. [27], we can consider software optimization using GPUs to generalize our results in a software environment. While it may not be as extremely fast as a single clock cycle in ASIC, it

is anticipated that our approach could generalize the implementation results in software environments with parallel computing.

- **Developing a General Entropy Measurement Formula:** Plans to study and develop a formula capable of measuring general entropy up to a certain loop across all Markov chain structures.

7 Conclusion

In our study, we proposed a single clock cycle permutation generation technique ARR which is essential for the shuffling countermeasure in PQC algorithms. We verified the attack complexity for an exhaustive search perspective using our newly proposed permutation verification method, *Identity Verification*. Also, we provided logical validity of the method through theoretical proofs. To demonstrate the relationships between each shuffled index, we experimentally presented second-order entropy. Both complexity of exhaustive search and second-order relations have been proven theoretically and experimentally to exhibit excellent scalability with small amount of randomness.

Based on our research result, it is possible to study subjects of theoretical and practical expansion. It is expected to provide significant utility to hardware designers and developers in the field of hardware & software co-design. Therefore, we hope that it can serve as a starting point for the proposal of even better structures. In future work, we intend to continue research to propose more efficient and higher-performing algorithms and better verification methods.

References

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

[2] FIPS, "Digital signature standard (dss)," FIPS PUB, Tech. Rep. 186-192, 2000.

[3] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. IEEE, 1994, pp. 124–134.

[4] NIST, "Nist round 4 submission," 2022, [Online] Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. Accessed on: Sep 2022.

[5] ——, "Fips 203 "module-lattice-based key-encapsulation mechanism standard"," https://doi.org/10.6028/NIST.FIPS.203.ipd, Tech. Rep.

[6] ——, "Fips 204 "module-lattice-based digital signature standard"," https://doi.org/10.6028/NIST.FIPS.204.ipd, Tech. Rep.

[7] ——, "Fips 205 "stateless hash-based digital signature standard"," https://doi.org/10.6028/NIST.FIPS.205.ipd, Tech. Rep.

[8] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science Business Media, 2008, vol. 31.

[9] N.-F. Standard, "Announcing the advanced encryption standard (aes)," Federal Information Processing Standards Publication, pp. 1–51, 2001.

[10] S. Tillich, C. Herbst, and S. Mangard, "Protecting aes software implementations on 32-bit processors against power analysis," in *Applied Cryptography and Network Security: 5th International Conference, ACNS 2007*. Zhuhai, China: Springer Berlin Heidelberg, 2007.

[11] R. A. Fisher and F. Yates, *Statistical tables for biological, agricultural and medical research*. Edinburgh and London, 1957.

[12] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[13] Z. Chen, Y. Ma, and J. Jing, "Low-cost shuffling countermeasures against side-channel attacks for ntt-based post-quantum cryptography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 322–326, 2022.

[14] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft analytical side-channel attacks," in *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security*. Springer Berlin Heidelberg, 2014, pp. 282–296.

[15] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America*. Springer International Publishing, 2019, pp. 130–149.

[16] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference*. Springer International Publishing, 2017, pp. 513–533.

[17] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, "Adapting belief propagation to counter shuffling of ntts," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 60–88, 2023.

[18] M. Dworkin, "Recommendation for block cipher modes of operation," NIST special publication, 2016, 38G.

[19] V. T. Hoang, B. Morris, and P. Rogaway, "An enciphering scheme based on a card shuffle," in *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference*. Springer Berlin Heidelberg, 2012, pp. 1–13.

[20] T. Ristenpart and S. Yilek, "The mix-and-cut shuffle: small-domain encryption secure against n queries," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference*. Springer Berlin Heidelberg, 2013, pp. 392–409.

[21] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *Topics in Cryptology—CT-RSA 2002: The Cryptographers' Track at the RSA Conference 2002*. Springer Berlin Heidelberg, 2002, pp. 114–130.

[22] A. Bacher, O. Bodini, A. Hollender, and J. Lumbroso, "Mergeshuffle: A very fast, parallel random permutation algorithm," *arXiv preprint arXiv:1508.03167*, 2015.

[23] J. Gustedt, "Engineering parallel in-place random generation of integer permutations," in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2008, pp. 129–141.

[24] J. Shun, Y. Gu, G. E. Blelloch, J. T. Fineman, and P. B. Gibbons, "Sequential random permutation, list contraction and tree contraction are highly parallel," in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 431–448.

[25] G. Cong and D. A. Bader, "An empirical analysis of parallel random permutation algorithms on smps." in *PDCS*, 2005, pp. 27–34.

[26] D. Langr, P. Tvrdík, T. Dytrych, and J. P. Draayer, "Algorithm 947: Paraperm—parallel generation of random permutations with mpi," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, pp. 1–26, 2014.

[27] R. Mitchell, D. Stokes, E. Frank, and G. Holmes, "Bandwidth-optimal random shuffling for gpus," *ACM Transactions on Parallel Computing*, vol. 9, no. 1, pp. 1–20, 2022.

[28] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, "Parallel random numbers: as easy as 1, 2, 3," in *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, 2011, pp. 1–12.

[29] V. Benes, "Optimal rearrangeable multistage connecting networks," *Bell System Technical Journal*, 1964.

[30] A. Shabani and B. Alizadeh, "Enhancing hardware trojan detection sensitivity using partition-based shuffling scheme," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 266–270, 2020.

[31] Lenfant, "Parallel permutations of data: A benes network control algorithm for frequently used permutations," *IEEE Transactions on Computers*, vol. 100, no. 7, pp. 637–647, 1978.

[32] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "Simon and speck: Block ciphers for the internet of things," Cryptology ePrint Archive, 2015.

[33] D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs," 2002, unpublished manuscript.

[34] J.-Y. Park, J.-W. Ju, W. Lee, B.-G. Kang, Y. Kachi, and K. Sakurai, "A statistical verification method of random permutations for hiding countermeasure against side-channel attacks," arXiv, 2311.08625, 2023. [Online]. Available: https://arxiv.org/abs/2311.08625

[35] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[36] NIST, "Data encryption standard (des)," FIPS PUB, Tech. Rep. 46-3, 1999.

[37] D. S. Dummit and R. M. Foote, *Abstract algebra*. Hoboken: Wiley, 2004, vol. 3, part 1 - Group Theory.
[38] D. A. Levin and Y. Peres, *Markov chains and mixing times*. American Mathematical Soc., 2017, vol. 107.
[39] J. L. W. V. Jensen, "Sur les fonctions convexes et les inégalités entre les valeurs moyennes," *Acta Mathematica*, vol. 30, no. 1, pp. 175–193, 1906.
[40] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference*. Springer Berlin Heidelberg, 1999, pp. 388–397.
[41] Microsoft learn, "Microsoft developer network," https://learn.microsoft.com/en-us/previous-versions/398ax69y(v=vs.140).
[42] J.-Y. Park, Y.-H. Moon, W. Lee, S.-H. Kim, and K. Sakurai, "A survey of polynomial multiplication with rsa-ecc coprocessors and implementations of nist pqc round3 kem algorithms in exynos2100," *IEEE Access*, vol. 10, pp. 2546–2563, 2021.

Jong-Yeon Park holds the position of Staff Engineer at Samsung Electronics, System LSI, and is currently enrolled as a Ph.D. candidate at Kyushu University. He received his Master's degree in Mathematics from Kookmin University in 2012. Subsequently, he served as a Researcher at the Electronics and Telecommunications Research Institute (ETRI) in Daejeon, South Korea, from 2012 to 2014. His professional experience also includes a Research Engineer at Korea Telecom's (KT) Convergence Laboratory in Seoul, South Korea, from 2015 to 2017. His research interests span a wide range of cryptographic topics, particularly those involving mathematical structures, secure algorithms, and Side-Channel Attacks (SCA).

Dongsoo Lee received the B.S degree in Electronic and Electrical Engineering from Sungkyunkwan University in 2012. He received the M.S degree in Electrical Engineering from KAIST in 2015. He joined the Telechips, Inc., in 2015, where he worked on system-on-chip and cryptographic IP design for conditional access system. He joined the Samsung Electronics, in 2020. From 2020, he focuses cryptographic accelerator design and robust design against side channel attack and fault injection attack. His research interests are IP design of cryptographic algorithm accelerator, and technique and methodologies for digital circuit resistant to side channel attack and fault injection attack.

Seonggyeom Kim received M.S. and Ph.D. degrees in Information Security from Korea University in 2018 and 2023, respectively. He currently serves as a staff engineer at Samsung Electronics, System LSI. His research interests encompass cryptanalysis, IP design of cryptographic algorithm accelerators, passive and active physical attacks, along with the corresponding countermeasures.

Wonil Lee received the B.S and M.S degrees in mathematics from Korea University in 1998 and 2000, respectively. He also received the Ph.D. degree in mathematics from Korea University in 2004. From 2004 to 2005, he was a researcher studying provable security of cryptographic hash function in Kyushu University, Fukuoka, Japan. Since 2005, he has worked as principal engineer for Smart card and Security industry in Samsung Electronic Ltd. His research interests are Security IC silicon security, Near Field Communication technology and system security for Mobile and IoT.

Bo-Gyeong Kang received the B.S Degrees in mathematics education from Seoul national university, south Korea, in 1999, and the M.A. Ph.D degrees in mathematics from Korea advanced institute of science and technology, KAIST in 2001 and 2005, respectively. She is currently a security group leader of Design Platform development team, Samsung S.LSI division. Her research interests include device security system with software and hardware, secure processor design integrated in SoC which satisfy security certification CC (Common Criteria), and new security algorithm implementation.

KOUICHI SAKURAI (Member, IEEE) received the B.S. degree in mathematics from the Faculty of Science, Kyushu University, in 1986, the M.S. degree in applied science and the Ph.D. degree in engineering from the Faculty of Engineering, Kyushu University, in 1988, and 1993, respectively. From 1988 to 1994, he was engaged in research and development on cryptography and information security at the Computer and Information Systems Laboratory, Mitsubishi Electric Corporation. Since 1994, he has been working with the Department of Computer Science, Kyushu University, as an Associate Professor, where he became a Full Professor, in 2002. He currently directs the Laboratory for Information Technology and Multimedia Security and is working with the CyberSecurity Center, Kyushu University. He is also with the Department of Advanced Security, Advanced Telecommunications Research Institute International. He has published about 450 academic articles in cryptography and cybersecurity.