A Compact and Parallel Swap-Based Shuffler based on butterfly Network and its complexity against Side Channel Analysis

JONG-YEON PARK, Samsung Electronics S.LSI, Korea. Rep
WONIL LEE, Samsung Electronics S.LSI, Korea. Rep
BO GYEONG KANG, Samsung Electronics S.LSI, Korea. Rep
IL-JONG SONG, Samsung Electronics S.LSI, Korea. Rep
JAEKEUN OH, Samsung Electronics S.LSI, Korea. Rep
KOUICHI SAKURAI, Kyushu University, Japan

A prominent countermeasure against side channel attacks, the *hiding countermeasure*, typically involves shuffling operations using a permutation algorithm. Especially in the era of Post-Quantum Cryptography, the importance of the hiding countermeasure is emphasized due to computational characteristics like those of lattice and code-based cryptography. In this context, swiftly and securely generating permutations has a critical impact on an algorithm's security and efficiency. The widely adopted Fisher-Yates shuffle, because of its high security and ease of implementation, is prevalent. However, it has a limitation of complexity $O(N)$ due to its sequential nature. In response, we propose a time-area trade-off swap algorithm, FSS, based on the Butterfly Network with only $\log(N)$ depth, $\log(N)$ works and $O(1)$ operation time in parallel. We will calculate the maximum gain that an attacker can achieve through butterfly operations with only $\log(N)$ depth from side channel analysis perspective. In particular, we will show that it is possible to derive a generalized formula of the attack complexity with higher-order side channel attacks for arbitrary input sizes through a fractal structure of the butterfly network. Furthermore, our research highlights the possibility of generating efficient and secure permutations utilizing a minimal amount of randomness.

Additional Key Words and Phrases: Post Quantum Cryptography, Permutation, Shuffling, Benes Network, Parallel Operation

ACM Reference Format:
Jong-Yeon Park, Wonil Lee, Bo Gyeong Kang, Il-jong Song, Jaekeun Oh, and Kouichi Sakurai. 2024. A Compact and Parallel Swap-Based Shuffler based on butterfly Network and its complexity against Side Channel Analysis. 1, 1 (July 2024), 28 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Background and Motivation

Shor's algorithm, a quantum computing breakthrough, presents a significant challenge to the security of traditional encryption methods such as RSA [43] and Elliptic Curve Cryptography (ECC) [14]. These conventional algorithms have long been the backbone of digital security, from online communications to financial transactions. However, Shor's algorithm can factor large integers and solve discrete logarithm problems in polynomial time with quantum computers, undermining the mathematical foundations on which RSA and ECC rely [47].

Authors' addresses: Jong-Yeon Park, jonyeon.park@samsung.com, Samsung Electronics S.LSI, Samsung Electornics Road 1-1, Hawsung, Kyeung-Ki Province, Korea. Rep, ; Wonil Lee, Samsung Electronics S.LSI, Samsung Electornics Road 1-1, Hawsung, Kyeung-Ki Province, Korea. Rep, ; Bo Gyeong Kang, Samsung Electronics S.LSI, Samsung Electornics Road 1-1, Hawsung, Kyeung-Ki Province, Korea. Rep, ; Il-jong Song, Samsung Electronics S.LSI, Samsung Electornics Road 1-1, Hawsung, Kyeung-Ki Province, Korea. Rep, ; Jaekeun Oh, Samsung Electronics S.LSI, Samsung Electornics Road 1-1, Hawsung, Kyeung-Ki Province, Korea. Rep, ; Kouichi Sakurai, Kyushu University, -, -, -, Japan.

-

XXXX-XXXX/2024/7-ART $0
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In response to this emerging threat, efforts have been made to develop and standardize Post-Quantum Cryptography (PQC) algorithms, with the aim of establishing new cryptographic standards [1]. The National Institute of Standards and Technology (NIST) has led this initiative, orchestrating a global, multi-phase competition to identify and standardize quantum-resistant cryptographic algorithms.

Side channel attack is an attack that discovers secret values by exploiting physical phenomena, such as electromagnetic signals and power consumption, that occur while cryptographic devices are in operation. Typically, based on the assumptions and methods of the attack, Differential Power Analysis (DPA) [20], Simple Power Analysis (SPA) [23], Correlation Power Analysis (CPA) [7], Timing attacks [21], and Template attacks [8], are known as the main techniques. Side channel analysis can be divided into two general categories of countermeasures: masking and hiding. Masking adds randomness to computations, making it impossible to model power and electromagnetic emissions, which are fundamental assumptions of side-channel analysis [24]. Countering higher-order side-channel attacks causes a significant increase in computational performance requirements and code size of the algorithm with only masking countermeasures. To overcome these issues, hiding countermeasures have been used as a low-cost supplementary method, making it more difficult to determine the temporal location of computations, thus complicating power analysis [52]. The most widely known hiding countermeasure is operation shuffling, which shuffles the presence of intermediate values that attackers seek to discover along the time domain [24]. Additionally, in the case of using multiple execution units simultaneously in hardware, hiding countermeasures can make it impossible for attackers to determine which execution unit is performing the target operation at any given time. For example, in the case of Advanced Encryption Standard (AES) [50], the mutual independence of the 16 S-box operations allows shuffling these 16 operations with a complexity of 16!.

A notable characteristic of PQC algorithms, such as lattice- and code-based algorithms [16, 32, 33], is their reliance on large vector operations. These operations are unique in that their sequence does not affect the outcome, paving the way for significant enhancements in countermeasures against side-channel attacks, going beyond traditional masking methods. The inherent flexibility in the operation sequence within PQC algorithms allows for the adoption of shuffling-based operation hiding as a primary countermeasure. This method involves randomizing the order of operations in the time domain or in the location within the physical area, significantly complicating the ability of an attacker to perform a successful side-channel analysis. In PQC algorithms, such as ML-DSA [33], can shuffle up to 256 operations, drastically increasing protection against such attacks. The advantage of applying hiding countermeasures in such PQC is that, from the perspective of an exhaustive search of the entire recombination of operations, it has an attack complexity of the total index of shuffled targets ($N!$) where N is the number of operations (indices). In the case of higher-order attacks, it would theoretically have a complexity of N combination r, assuming the shuffling is completely random where r is the order. From this perspective, there is a significant difference between having $N = 16$ and $N = 32$. When $N = 16$, $16! \approx 2^{44}$, whereas $32! \approx 2^{117}$. Thus, in practical terms, the difference in complexity for an exhaustive search is immense.

1.2 Evaluating shuffling security in the context of side-channel analysis

When the object being shuffled achieves complete complexity, i.e., it is randomly shuffled with uniform probability across all possible $N!$ cases, it functions as an ideal countermeasure. However, if the shuffler is biased, its security from the perspective of side-channel analysis must be evaluated in the following ways:

- **Shuffling as a countermeasure against single trace attacks (profiling or non profiling)** [40, 41, 49, 55]: If an attacker can fully analyze the target using a single trace attack or similar methods, the extent to which the permutation is shuffled needs to be evaluated. Ideally, if the index size is assumed to be N, it would be shuffled with a uniform probability distribution across $N!$ possibilities. However, $O(N!)$ represents an unnecessarily high complexity, so a practical compromise might be ideal for actual environment of industries. In this case, the time complexity required to recombine the output sequence should be measured, which

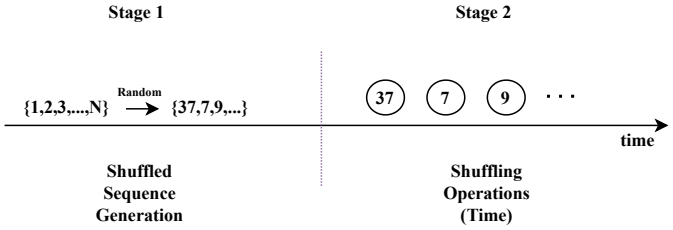
can be quantified using Shannon Entropy with an statistical model. These measurements are necessary for validating the effectiveness of countermeasures against revealing the key through recombination with all shuffled key buffers.

- **Shuffling as a countermeasure against DPA attacks** [25, 52, 54]: When an attacker performs a DPA analysis, the complexity of the attack when shuffling is implemented as a countermeasure should be measured. For a primary DPA attack, the probability of a specific operation occurring at a specific time (or location) must be calculated. Ideally, if the countermeasure involves mixing N operations, the ideal probability would be $1/N$. If a secondary DPA analysis is necessary, the attack would target two points with the same masking to determine any correlation between them after shuffling. This requires checking the computational complexity for $\binom{N}{2}$ pairs of indices. Similarly, to determine the complexity of an attack corresponding to an h-th order attack, the distribution of states for $\binom{N}{h}$ must be analyzed to calculate the Shannon entropy. This measurements are necessary for validating the effectiveness of countermeasure against higher order DPA attacks.

- **Shuffling as a countermeasure against attacks using the relationship of each index** [38, 39]: There is an attack that is not in the traditional style of SPA attacks, nor simply a profiling attack to find specific values. Instead, this attack solves equations to find the key using various additional statistical techniques on top of conventional profiling attacks. In such cases, the relationships between each index are more important than the overall mixing of entropy. In this attack as well, it is necessary to measure entropy based on the same criteria as applying shuffling in higher-order DPA.

1.3 An implementation perspective for operation shuffling

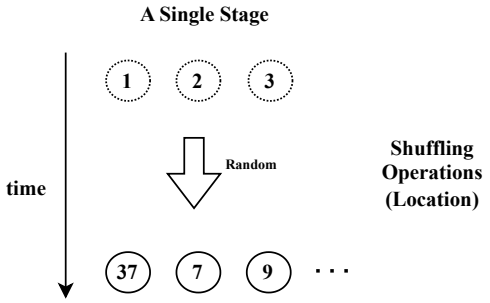
Fig. 1. Sequence Generation and Shuffling Operation: Sequence and Shuffling (S-and-S)



From an implementation perspective, shuffling can be broadly divided into three areas. The first is generating a shuffling sequence and determining the order of operations based on the generated sequence, namely *Sequence and Shuffling (S-and-S)*; see Figure 1. This is the most commonly used method and is intuitively straightforward. Since all cryptographic algorithms implemented in software cannot be executed in parallel, this strategy can be considered the only viable method [13, 52]. This method can be used when you want to arbitrarily rearrange the chronological order of all sequentially performed operations, regardless of whether the main operations are implemented in hardware or software.

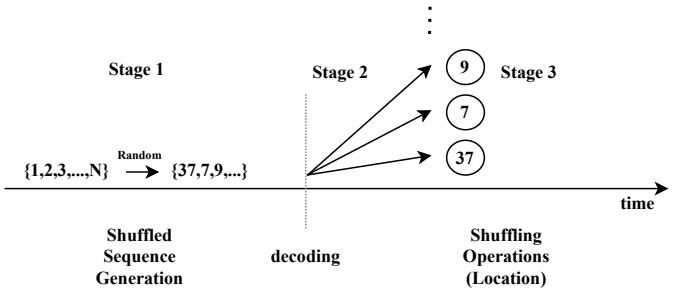
The second method involves shuffling the operations by randomly changing their positions. This approach is generally used when the entire implementation is composed of hardware. An intuitive illustration of this

Fig. 2. Merging Sequence Generation and Shuffling Operation in Parallel: Shuffling at Once (S-At-Once)



can be seen in Figure 2, namely *Shuffling at Once (S-at-Once)*. In this case, a separate sequence for performing the permutation is not generated. Instead, the focus is on the allocation of positions where each operation is performed. This is generally implemented using methods such as switching networks [9, 56].

Fig. 3. Sequence Generation and then Shuffling Operation in Parallel: Sequence and Allocation (S-and-A)



However, even in situations where all operations are parallelized and the entire system is implemented in hardware, a shuffling sequence can be generated in advance to redistribute the positions of the operations, namely *Sequence and Allocation (S-and-A)*. For this redistribution, Stage 2 in Figure 3 is necessary, and a step must be taken to convert values to positions using a multiplexer or demultiplexer.

Despite various implementation methods, high-speed shuffling generation is fundamentally important for both changing the positions of operations and rearranging their chronological order. In conclusion, the shuffling generator is significant in its own right for hardware, software, parallel, and sequential shuffling. Bayrak et al. [3] referred to this study as an architecture-independent instruction shuffler." In this paper, we expect that the generated sequence will have sufficient complexity and that the shuffling generator will have a fully parallelized hardware structure for high-speed performance. This would minimize operational delays wherever the shuffling is applied. For example, in the case of an algorithm implemented entirely in software, if the shuffling generator can be generated in hardware within one clock cycle [35], all intermediate shuffling can be applied with minimal operation delay.

1.4 Existing works and its Limitations

1.4.1 Fisher-Yates Shuffling. Shuffling techniques, which are methods typically used to generate sequences through permutation generation algorithms and use the generated indices for shuffling operations, have various applications. Among permutation generation algorithms, also known as shuffling sequence generation algorithms, Fisher-Yates (FY) shuffling is the most widely used [13, 19]. The popularity of FY shuffling comes from its ease of implementation and its ability to produce a complete permutation. However, this algorithm faces several drawbacks [34]: first, it must be executed sequentially, which limits the potential for process acceleration; second, it is susceptible to various misuses; and third, it demands an excessive number of random numbers. To overcome these problems, several algorithms have been studied that modify FY shuffling to enable parallel computation. However, this research has structural limitations in performing the computations completely in parallel [2, 10, 17, 22, 48].

To address these challenges, more research is necessary. Firstly, a high-speed shuffling generation algorithm capable of parallel processing is needed. Secondly, it is essential to generate random numbers of a manageable size to maintain the required complexity and to control the size of the needed random numbers.

1.4.2 Side-Channel Attacks and Countermeasures on PQC. Number Theoretic Transform (NTT) operations, notable for their shuffleability, serve as an efficient mechanism for operating lattice-based cryptographic computations. Polynomial operations, unlike integer operations, do not carry over, inherently allowing parallel processing in addition, subtraction, and multiplication. NTT operations can maximize this attribute of parallel processing, enabling operation and recombination in a format that requires only $\log(N)$ depth and $N/2$ butterfly units, regardless of the order.

Ravi et al. [41] categorize these operations as Coarse-Full-Shuffled NTT, Coarse-In-Group-Shuffled NTT, and Fine-Shuffled NTT, demonstrating the results using FY shuffling in ML-DSA [33] and ML-KEM [32]. The division of shuffling methods is based on efficiency considerations: whether all butterfly units are shuffled or only a part is grouped and shuffled, leading to a trade-off between performance and security. Fine-Shuffled NTT involves conditionally swapping two operations within a single Butterfly unit, proposed as an additional method to counteract attacks like those identified by Pessl et al. [37].

Chen et al. [9] proposed an efficient method for implementing a shuffling index in NTT implementations based on FPGAs. Their approach does not use full permutation generation methods, such as FY shuffling, but employs a one-time rotation of the NTT index through the modular addition of $N/2$. This achieves maximal shuffling complexity against first-order DPA attacks. Integrating only fine shuffling using Ravi's conditional swap, this method maintains high efficiency, with an overall shuffling complexity of $N/2 \times 2^{N/2}$. For the ML-DSA and ML-KEM standards with $N = 128$, this approach makes attacks virtually impossible in simple forms of single trace template attacks. Although this method marks a significant achievement in the application of highly efficient shuffling to NTT operations with about a 10% performance delay, its structural simplicity in shuffling could potentially make it more susceptible to attacks [38, 39].

Zijlstra et al. [56] also proposed an FPGA implementation of shuffled NTT, with the distinctive feature of utilizing the Benes network to optimize both the complexity and performance of shuffling. They enhanced the performance of extracting random bits by connecting an Linear Feedback Shift Register (LFSR) to the Network. However, their research does not verify the complexity of this network.

1.4.3 Small Domain Encryption and Format-Preserving Encryption. Encryption for small domains fundamentally involves permutations. The main focus in this area is Format-Preserving Encryption (FPE), where research has primarily focused on encryption schemes that manage input sizes flexibly. Before FFX standardization [12], various approaches were explored, including encryption methods similar to card shuffling, the use of block

ciphers [18, 42], fixed ciphers based on table lookups [6], and techniques that adaptively manage input and output sizes through the operational modes of block ciphers [12].

Furthermore, for particularly small permutations used in shuffling, specifically those where the total number of indices is at most 2^{12}; existing FPE algorithms, to our knowledge, require an excessive number of operations. This observation identifies a small-scale permutation efficiency gap, indicating a need for more streamlined solutions in the FPE domain. In the case of standard FFX mode, a secure encryption algorithm is used as the base, and AES is generally employed. However, the performance overhead of the FFX mode with AES is substantially higher than that of direct encryption and decryption on small domains, making it impractically slow for such applications. Furthermore, in other methods, when the total index size (plaintext space) N is very small, the number of rounds required is quite large. In the minimal case, the Swap-or-not method requires at least $10\log(N)$ rounds [18]. Other methods typically require $poly(\log(N))$ rounds [6, 42].

The reason for this inefficiency is that the purpose of Format-Preserving Encryption (FPE) itself is based on achieving Chosen Ciphertext Attack (CCA) security. This involves an oracle model where the attacker can monitor both intermediate inputs and outputs. This scenario is fundamentally different from the entropy considerations in side-channel analysis. However, the provable security in FPE has been narrowly focused on the distinguishability from a random permutation within a certain query size. This focus ensures encryption and decryption security but does not consider the overall entropy or complexity of permutations. Our study aims to delve into the full scope of permutation entropy and complexity, diverging from the current security paradigms in FPE.

1.4.4 Low Performance Partially Parallel Permutations. Investigating the generation of random permutations in parallel presents a unique line of inquiry and has been tackled using various methods [2, 10, 17, 22, 36, 48]. This research area has two common characteristics. First, it seeks to blend parallelism with sequential elements, thus not fully leveraging the potential for hardware efficiency through complete parallel processing. Second, these methods aim to produce entirely uniform permutations, equating to the complexity of $N!$. This implies that stopping the algorithm partway results in some indices remaining unshuffled or their complexity unproven, thereby revealing a trade-off between complexity and efficiency. As such, these methods are not ideal for security applications due to their inefficiency and the excessive demand for random numbers.

Approaches also exist that utilize block ciphers, which are permutation-based parallel algorithms operated on graphics processing units (GPUs) with bijection through Feistel networks [26, 44]. These algorithms share similarities with our fully parallel random shuffling approach. This research indicates that block bijection can generate uniform random permutations that meet their own statistical test criteria; however, it does not accurately reflect complexity. Moreover, because their research does not calculate the actual complexity of permutations, passing their tests does not equate to having the attack complexity of a factorial.

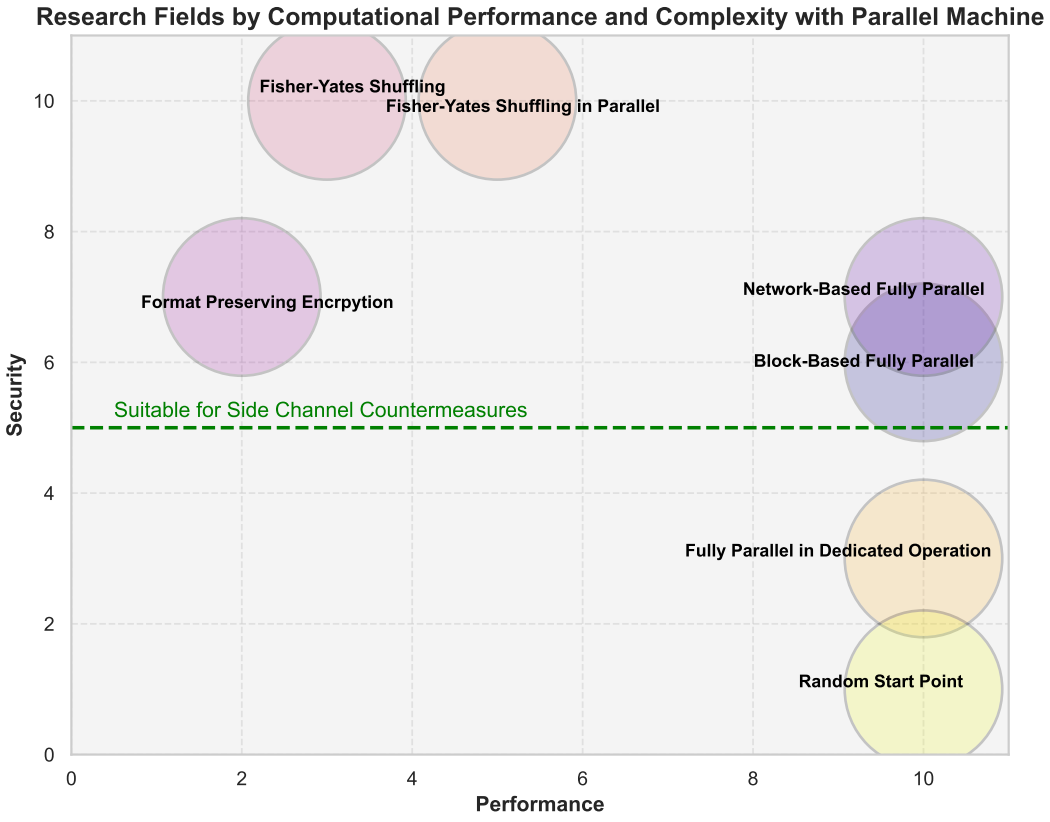
1.4.5 Block and Network-based Fully Parallel Permutations. The research that is most closely related to ours, with a similar objective, is identified as ARR [35]. This study utilizes the concept of block ciphers, employing only addition and rotation to iteratively execute round functions, thereby generating permutation sequences with sufficient complexity within a single clock cycle. In addition, it showcases the complexity of second-order analysis that can be directly applied to proving security against side-channel attacks. However, this research has two limitations. Firstly, it utilized the innovative idea of identity verification to calculate the overall complexity of the permutation, but suggested that statistical experiments are mandatory to predict the total complexity. Secondly, the second-order analysis, which evaluated the mixing of two positions, was not theoretically proven but demonstrated through experimental evidence.

Shuffling can alternatively be implemented using Routing Networks such as the Benes network [4, 45] (a 2-depth symmetric butterfly network), with minor variations. The advantages of the network-based approach include the ability to perform parallel operations at high speeds and generate complete permutations.

One key feature is the equivalence of the butterfly network with Thorp shuffling, which is known to randomly cross two card decks. This characteristic has inspired various studies. Ben Morris conducted extensive research on the efficacy of Thorp shuffling (butterfly network permutation) in generating all permutations, indicating roughly the total variation distance (TVD) for q inputs in $poly(\log(N))$ [27–29].

Meanwhile, Czumaj [11] calculated the complexity of the network at a depth $\log^2(N)$ through coupling methods in non-Markovian settings for the performance of the butterfly network. In contrast, Gelman et al. [15] calculated the upper bound of the attacker's gain that can be achieved through queries q in the Benes network. Although the main purpose of this research differs from ours, the problems addressed are very similar, speculating on the attacker's gain obtainable through the Benes network using the total variation distance as a measure.

Fig. 4. Research Fields by Computational Performance and Complexity with Parallel Machine; Fisher-Yates Shuffling [13, 19], Fisher-Yates Shuffling in Parallel [2, 10, 17, 22, 48], Format Preserving Encryption [12], Network-Based Fully Parallel structure [15], Block-Based Fully Parallel Structure [26, 44], Fully Parallel in Dedicated Operation [9], and Random Start Point [53]



1.5 Challenging Issues

Therefore, when we synthesize the forms of these studies, we need a study that can show high performance while having the complexity of permutations that are useful as hiding countermeasures. The x-axis of Figure 4 represents performance. Performance varies greatly depending on implementation methods and practical conditions, but we show performance in terms of computational complexity. The y-axis is the level of security. The most complete form has a computational complexity of $N!$ such as FY shuffling. Since a sufficiently secure permutation for side channel analysis does not need to have full complexity of $N!$, there is a need for discussion about this, and we marked reasonable security levels with a green line. This line includes the guessing entropy of recombining the entire shuffling and the higher-order analysis.

To develop an algorithm capable of generating permutations fully in parallel and at extremely high speeds, our aim is to apply the butterfly network with just $\log(N)$-depth and $\log(N)$-works, analyzing its corresponding attack complexity from the perspective of countermeasures against side-channel attacks. This objective aligns closely with the research conducted by Park et al. [35], which mentions network-based methods but lacks a theoretical proof of security or high-order index shuffling in fully permutation scenarios.

Research on the complexity of fully parallel permutations using the butterfly network has progressed in one of the following categories:

(1) Cryptographically determining a specific number of queries and calculating the limit of gain an attacker could achieve, or Calculating the number of rounds required to meet the statistical boundary for a certain probability of attack [11, 27, 30].
(2) Only the full-depth Benes network has been studied [15], without research on the $\log(N)$-depth butterfly method.
(3) The research only implemented shuffling using the Benes network and conducted Differential Power Analysis (DPA) experiments on side-channel analysis, but did not analyze the actual complexity of the implementation [3].
(4) The feasibility of side-channel attacks has not been analyzed, or it was not even a key research topic, but only partially and solely in the second order, in an experimental context [35].

1.6 Our Contributions

Our research focuses on the butterfly network at depth $\log(N)$, which we will refer to as Fixed-Swap Shuffling (FSS). This paper contributes the following:

(1) It proves the attack complexity of fully permutations, serving as a countermeasure against Single Trace side-channel attacks.
(2) It demonstrates the attack complexity for higher-order permutations, showing the complexity of how indices originating from positions q are determined after permutation.
(3) Finally, we compare the q-th order complexity between the result of the computation of the $\log(N)$-depth butterfly network derived by our computation and the computation result of worst-case probability of E. Gelman et al. [15] with q queries.

Furthermore, this paper will not cover the methodologies for generating random numbers, which is a very crucial factor in creating permutations. There are various methods for generating random numbers, each with its own performance, complexity, and impact on the final shuffling sequence, making it a topic extensive enough to require separate research. Therefore, we assume that random numbers are provided in a completely uniform. We will only proceed with the basic premise that using fewer random numbers can benefit overall computational performance.

The paper is organized as follows. Section 2 explains the preliminary knowledge required to understand the paper. Section 3 details how the FSS algorithm operates. Section 4 discusses the security of FSS from the

perspective of using side channel attack countermeasures. Section 5 discusses several useful remarks obtainable under the Benes network to compare between our result and the full-depth Benes network with the computed entropy from q-set-wise independence. Finally, Section 6 will discuss the limitation and future work and concludes the paper with discussions related to cryptographic applications.

2 PREREQUISITES

2.1 Notations and Definitions

The $\mathbb{S}_N$ represents the symmetric group on the set $[N] = \{0, 1, 2, 3, \ldots, (N-1)\}$. A permutation is a one-to-one correspondence function. Moreover, we limit the domain to the set $[N]$. Therefore, by the definition of a permutation, the output range is also $[N]$.

$$\mathsf{Perm} : K \times [N] \rightarrow [N]$$

Refers to a family of permutations that transform integers. Hence, Perm has the property that, for each $k \in K$, the projection $\mathsf{Perm}(k, \cdot)$ is a permutation on $[N]$. The realizations of Perm are represented as ϕ, δ, or FSS. The set K denotes a key set that determines the diversity of the permutation with the given method Perm. Therefore, $\mathsf{Perm}(K, x) = y$ represents a set of *pseudo-random permutations* determined by a key set K, including random numbers $\{k_i\}$. Mentioning $\mathsf{Perm}(x)$ without specifying K implies that k is randomly chosen in uniformly distributed K. In this context, the term *random permutation* refers to a pseudo-random permutation. 'I' is the identity permutation, where $I(x) = x$ for all $x \in [N]$, making $\mathsf{Perm}(k, \cdot)$ possibly the identity permutation chosen k.

For the set $[N]$, $\binom{[N]}{q}$ denotes the set of all subsets q of $[N]$, that is, $\{A \subseteq [N] \mid |A| = q\}$. In this context, N is a power of 2, which signifies $|[N]| = N = 2^n$ with $n = \log_2(N)$. Assuming X represents a subset in the power set (sets of subsets) of the ordered set $[N]$, Φ_X refers to a $|X| \times |X|$ transition matrix of a Markov chain, where $0 \leq (\Phi_X)_{i,j} \leq 1$ indicates the probability that the i-th element changes to the j-th after a stochastic process. $[\pi_X]$ signifies the set of all elements $((\Phi_X)_{i,j})$ within the matrix. For example, with $X = \{\{0\}, \{1\}, \{2\}, \ldots, \{N-1\}\}$, which is a set of single elements, π_X is a transition matrix $N \times N$. In such a scenario, the set is termed a *1st-order set*, $X = \binom{[N]}{1}$. Similarly, a *second-order set* is defined with the set of all pairs of elements, e.g. $X = \{\{0, 1\}, \{1, 2\}, \{2, 3\}, \ldots, \{N-1, 0\}\} = \binom{[N]}{2}$, with $|X| = \binom{N}{2}$. When X equals $\mathbb{S}_N$, then π_X is $N! \times N!$ matrix.

Definition 2.1. The *q-th order (attack) complexity* on $X = \binom{[N]}{q}$ refers to the worst-case security for q inputs and outputs in $\binom{[N]}{q}$, calculated as $\frac{1}{max([\pi_X])}$. The term $\max([\pi_X])$ is defined as the *q-th order (attack) probability*, which is the highest probability value for the inputs of $\binom{[N]}{q}$ to the outputs.

Definition 2.1 outlines the worst-case security considerations for a q-th order attack. For example, in the second-order attack scenario for a masked implementation, an attacker would choose two points with identical masking and use the power values at these points for the attack. Success in this context depends on whether there are precisely two operations at the chosen points or they are exactly on opposite points. Therefore, the maximum probability $\max([\pi_X])$ emerges as the highest value among the probabilities that an attacker selects two points, and operations occur at those selected points for the attacker. The *q-th order attack complexity* and the *q-th order attack probability* share an inverse relationship, offering a logarithmic perspective of opposition. While these concepts are used interchangeably depending on the context, they are effectively reflections of the same principle since one's value directly influences the other's.

Definition 2.2. The *q-th Order Probability of input x_q for a permutation* Perm is defined as probability of a random variable X which is an output of a shuffled index $\mathsf{Perm}(x_q)$ with input $x_q \in [N]$ and $q \leq N$ under values $\mathsf{Perm}(x_1), \mathsf{Perm}(x_2), \mathsf{Perm}(x_3), \ldots, \mathsf{Perm}(x_{q-1})$ have already determined, denoted as $\Gamma_q^{x_q} = \Gamma_q^{x_q|x_1,x_2,\ldots,x_{q-1}}$.

Definition 2.2 introduces a novel metric for assessing the randomness under a q-th order attack for a specific permutation. For example, considering a uniformly random permutation across all indices n, the first-order probability at any $x_1, y \in [N]$ would be $\Gamma_1^{x_1} = \frac{1}{n}$ for all x_1, and the second-order probability at any input x_2 (and x_1 is fixed input) would be $\Gamma_2^{x_2} = \frac{1}{n-1}$ for all x_2. This methodology, $\Gamma_q^{x_q}$, will be employed to determine the q-th order attack complexity of the proposed permutation for a side channel attack aspect. It is important to note that the definition of $\Gamma_q^{x_q}$ itself is influenced by results of the inputs from x_1 to x_{q-1}. If the random permutation is biased, it will be affected by the previous results according to the prior input values such as x_1, x_2, x_3, etc. Therefore, even for the same permutation, this value will not be identical and it will also form a probability distribution. The maximum values of the product of all probability values, is intuitively the same as the q-th order attack probability. This will be discussed in Section 4.3.

A *transposition* (A, B) implies that A transforms to B, and B to A. If two transpositions are disjoint, for example, (A, B) and (C, D), where A, B, C, and D are distinct elements, then their composition is commutative, which implies that $(A, B) \circ (C, D) = (C, D) \circ (A, B)$. This is succinctly represented as $(A, B)(C, D)$.

The probability of an event E occurring is denoted as $\Pr(E)$. Furthermore, all references to 'log' in this paper signify the base-2 logarithm, used to calculate the entropy as $H(X)$ with a random variable X [46]. The entropy is calculated as follows:

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

Shannon entropy, $H(X)$, represents the computational time complexity to guess secret information on a scale of log, such as $2^{H(X)}$. Additionally, $H_{min}(X)$ (*minimum entropy*) refers to the inverse of the highest probability of a random function; p^{-1}, where p is the maximum probability. *Ideal entropy* denotes the maximum theoretical entropy for a given random variable. In a uniform distribution, the following fact is naturally accepted.

Fact 2.3. If the random variable X has the uniform distribution, then the Shannon entropy and the minimum entropy are identical.

By definition, the value of Shannon entropy is always greater than the value of minimum entropy, and by definition, the value of minimum entropy aligns with the worst-case attack complexity in Definition 2.1.

The total variation distance (TVD) is a metric to quantify the difference between two probability distributions [5]. TVD between two probability distributions P and Q in a probability space Ω is defined as:

$$\|P - Q\| = \sup_{A \subseteq \Omega} |P(A) - Q(A)| \tag{2}$$

Alternatively, it can be expressed as half of the L^1-norm of the difference between the two distributions:

$$\|P - Q\| = \frac{1}{2} \sum_{x \in \Omega} |P(x) - Q(x)| \tag{3}$$

TVD ranges from 0 to 1, where 0 indicates identical distributions and 1 denotes maximally different distributions. It provides a clear and interpretable measure of the discrepancy between two probability distributions.

Fig. 5. 8×8 Benes Network 5-depth (3-depth butterfly network up to 3-depth) - Switch Description

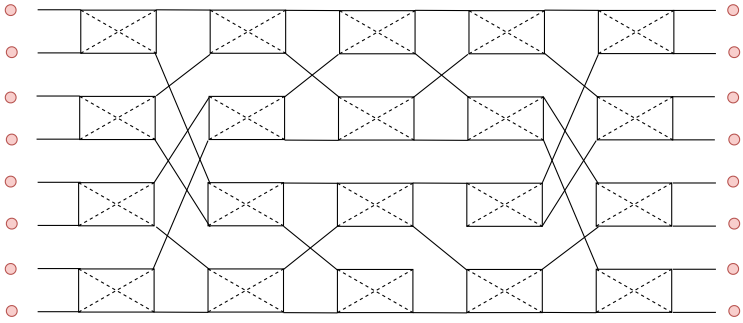


Fig. 6. (2×2) Switch

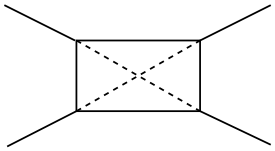
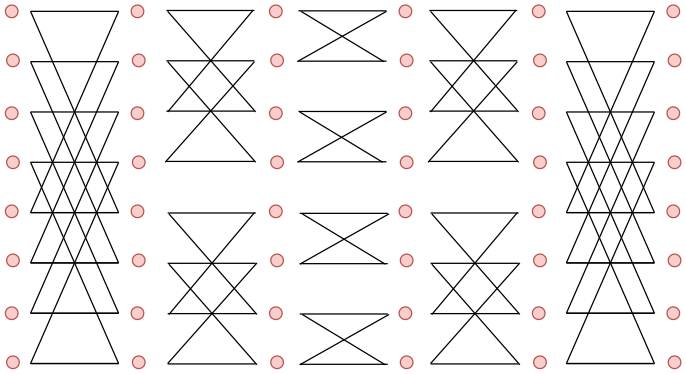


Fig. 7. 8×8 Benes Network 5-depth (3-depth butterfly network up to 3-depth) - Location Swap Description



2.2 Routing Network and Thorp Shuffling

Routing networks are specialized networks designed to route data through a series of nodes and pathways. They are fundamental in Telecommunications, Parallel Computing, Data Center Networks, etc. Routing networks play a crucial role in the field of computer science and telecommunications, where they are used to efficiently manage the flow of data between various points [51].

One prominent example of a routing network is the Benes Network [4]. This routing structure can be described as two methods: a switching system that uses 2×2 switches (Figures 5 and 6), and a method that employs value-swap operations; see Figure 7. These two methods ultimately represent the same system. We are interested in the location swap description since it can be presented in general algorithm form and expanded the same small unit to full size. Only the $\log(N)$-depth Benes network is called a butterfly network. On the other hand, the $2 \times \log(N) - 1$ butterfly network is considered the Benes network.

Thorp shuffling is another view for the butterfly network, which is only $\log(N)$-depth butterfly network[27–29]. Thorp shuffling is a simple model for random riffle shuffle in card games. One first cuts a deck of cards in half and then starts dropping the cards from either the left or right hand as with an ordinary shuffle. In each step, the left or right card with a probability of $\frac{1}{2}$ is chosen. After the $\log(N)$-depth butterfly network, it is exactly the same result for Thorp shuffling as a card deck and its locations.

3 FIXED-SWAP SHUFFLE (FSS): REARTICULATED ALGORITHM OF BUTTERFLY NETWORK

3.1 Basic Structure: Algorithmic Description of Butterfly Network

Algorithms 1 and 2 represent the Fixed Swap Shuffling (FSS) algorithm which is the same but clear to describes $\log(N)$-depth butterfly network and Thorp Shuffling , with the only difference between the two algorithms being that Algorithm 1 generalized the Deterministic Pseudo random function (DPRF) which is determined by inputs only, while Algorithm 2 specifies DPRF by generating random bits first and then selecting the positions of the random bits ($R = r_0 r_1 r_2 \ldots r_{((N/2)\times n)-1}$).

$$\mathsf{DPRF} : \mathbb{Z} \rightarrow \{0, 1\}$$

Each approach has its advantages and disadvantages. The first approach can be seen as a generalization of the second approach, which is advantageous in that it can be used as a general encryption scheme for only certain inputs to be encrypted if DPRF is assumed to be random for an input of arbitrary size, for example, a 128-bit input. It has the different purpose from permuted sequence generator. If we design a 128-bit permutation using the second method, the length of the random number to be generated becomes infinitely long. Only a bit stream of length $2^{128}/2$ is required for the first depth. However, if the algorithm is proven to be secure for a given attack size, there is no need to use a method such as Algorithm 2. In addition, since output values are not required for all input sizes, it is better to compute the PRF algorithm than to produce specific bits in advance.

However, if it is assumed that output values are needed for all inputs, applying DPRF to all switches determined by one bit would be quite inefficient, and it would be more advantageous to generate each bit (each switch) in advance. Each random bit generated can serve as DPRF. Especially, for our aim of shuffle indices for side channel attack with small domain permutation, Algorithm 2 may be a better choice. This method can work as a generator for shuffled sequences.

Therefore, Algorithm 3 shows a routine that generates output values by performing permutation algorithms using FSS on all indices. The generalized form is to use Algorithm 1 (DPRF), but this part can generally be considered advantageous to use Algorithm 2's random bit when the input size is small. The time complexity of this operations is $(O)(N)$. As a sequence generator implemented in hardware, this operation can be parallelized with a complexity of $c \times O(1)$, where c is the execution time of Algorithm 1 or 2.

3.2 Transposition Representation for FSS

In order to compute the guessing entropy of FSS, we show that FSS can be represented through compositions of disjoint transpositions. For example, consider the 8×8 network illustrated in Figure 7. The initial layer generates swap operations through transpositions $(0, 4)$, $(1, 5)$, $(2, 6)$, $(3, 7)$. Thus, FSS consists of $\log(N)$ rounds (depth) of

Algorithm 1 Fixed-Swap Shuffle (FSS) - Permutation FSS(DPRF, x) with Deterministic Pseudo Random Function (DPRF) for General Input/Output Model, $O(\log(N))$

Input: An integer $x \in [N]$ such that $N = 2^n$ and DPRF
Output: FSS(DPRF, x) $\in [N]$

for $j = 0$ to $n - 1$ **do**
  $s \leftarrow$ left rotation j bits of x
  $t \leftarrow$ right $(n-1)$ bits of s /*LSB (n-1) bits (without MSB)*/
  $t \leftarrow \mathrm{DPRF}(t \| j)$ /*t is 1 bit output of PRF */
  if t equals 0 **then**
    $x \leftarrow x$ /*No operation*/
  else
    $x \leftarrow x \oplus 2^{(n-1)-j}$ /*1 bit Exclusive-OR*/
  end if
end for
return x

Algorithm 2 Fixed-Swap Shuffle (FSS) - Permutation FSS(R, x) with deterministic random bits with length $(N/2) \times \log(N)$ for Small Domain Permutation Model, $O(\log(N))$-Time

Input: An integer $x \in [N]$ such that $N = 2^n$ and random bit stream $R = r_0 r_1 r_2 \ldots r_{((N/2)\times n)-1}$
Output: FSS(R, x) $\in [N]$

for $j = 0$ to $n - 1$ **do**
  $s \leftarrow$ left rotation j bits of x
  $t \leftarrow$ right $(n-1)$ bits of x /*LSB (n-1) bits (without MSB)*/
  $t \leftarrow R_{t+(N/2)\times j}$ /*On the fly PRF with Random Bits*/
  if t equals 0 **then**
    $x \leftarrow x$ /*No operation*/
  else
    $x \leftarrow x \oplus 2^{(n-1)-j}$ /*1 bit Exclusive-OR*/
  end if
end for
return x

Algorithm 3 Shuffling Sequence Generator with Fixed-Swap Shuffle - FSS(DPRF, $\cdot$), $O(N)$-Time

Input: DPRF
Output: A random sequence $Y = \{y_0, y_1, y_2, \ldots, y_{(N-1)}\}$, $y_i \neq y_j$ where $i \neq j$, denote FSS(DPRF, $\cdot$)

for $i = 0$ to $N - 1$ **do**
  $y_i \leftarrow$ FSS(DPRF, i) /*Can be operated in parallel for each y_i*/
end for
return Y

such disjoint transpositions. We introduce the concept as a given and proceed without further elaboration. This concept will be foundational for the later proved significance of Proposition 4.1.

Define T_L as a set of disjoint transpositions for layer L, that is, $T_L = \{t_{L,0}, t_{L,1}, \ldots, t_{L,m-1}\}$. For example, $T_0 = \{(0,4), (1,5), (2,6), (3,7)\}$, in the 8×8 network. Consequently, a selected random composition of transpositions

from T_L (Layer L), denoted by $\sigma_L = t_{L,k_0} t_{L,k_1} \ldots$, does not depend on the order due to their disjointness. FSS selects a random permutation through these disjoint transpositions, constructing sequences such as $\sigma_{\log(N)-1} \circ \ldots \circ \sigma_1 \circ \sigma_0$, where σ_j encompasses all elements of T_j. Thus, we can formally describe this fact as Fact 3.1.

Fact 3.1. The algorithm $\mathsf{FSS}(k_i, \cdot)$ operates through the function composition $\sigma_{\log(N)-1} \circ \ldots \circ \sigma_1 \circ \sigma_0$, with each σ_i representing a layer composed of disjoint transpositions. Notably, the transpositions within each individual σ_j are not disjoint.

Fact 3.2. Within the FSS, for any distinct layers z and y_j, where $z, y_j \leq \log(N) - 1$ and $\sigma_z \neq I$, it holds that σ_z is always distinct from any composition $\sigma_{y_1} \circ \sigma_{y_2} \circ \ldots \circ \sigma_{y_\tau}$, where each y_i may not be distinct, utilizing the unique transposition notation for each layer.

Fact 3.2 is intuitively true, so we omit the proof of this statement. Each step is a bit flip of the internal values of FSS. Given that no matter how you change the bits in different positions, you cannot change the value of a specific bit. Thus, Fact 3.2 can be understood as correct.

3.3 Reduced Random Source FSS (R-FSS)

Algorithm 2 is described based on usage with full-length random bits. If we consider the reduced length of random bits, there is a trade-off between security, performance of generating random bits, and size of buffer storing random bits. For example, one may generate random bits for only the first layer, and use these random bits for the other layers of all depth.

4 SECURITY

4.1 Entropy of Full Permutation

If we assume that an attacker can find all sensitive values through methods like template attacks, shuffling can be considered as a countermeasure. To calculate the time required to recombine all values, it would be appropriate to measure the guessing entropy of the recombination by considering the permutation as a probability distribution. Instead of calculating the Shannon entropy of our FSS, we will calculate its upper bound in Proposition 4.1, the minimum entropy, to estimate the bounds of Shannon entropy. This aims to provide developers with a calculable safe bound as a countermeasure against single trace attacks. Additionally, the effect of reducing the used random values on the total number of recombination cases will be provided in Corollary 4.2.

Proposition 4.1. For FSS with $N = 2^n$ inputs, the entropy equals the entropy of a uniformly distributed random bit stream of length $((n-1) \times N/2)$.

Proof. To demonstrate that two identical permutations generated by FSS, given the random bit streams R_1 and R_2 of $((n-1) \times N/2)$ bits, result in the same sequence of transpositions, consider the following. Let $\mathsf{FSS}(R_1, \cdot) = \sigma_{y_0} \circ \sigma_{y_1} \circ \ldots$ and $\mathsf{FSS}(R_2, \cdot) = \sigma_{z_0} \circ \sigma_{z_1} \circ \ldots$, where σ_{y_i} and σ_{z_i} represent the composition of the disjoint transpositions of i-th layer, and $\mathsf{FSS}(R_1, \cdot) = \mathsf{FSS}(R_2, \cdot)$. Thus, we can write:

$$\sigma_{y_0} \circ \sigma_{y_1} \circ \ldots = \sigma_{z_0} \circ \sigma_{z_1} \circ \ldots \tag{4}$$

Rearranging terms gives:

$$(\sigma_{z_0} \circ \sigma_{y_0}) \circ \sigma_{y_1} \circ \ldots = \sigma_{z_1} \circ \sigma_{z_2} \circ \ldots \tag{5}$$

Further simplification leads to

$$(\sigma_{z_0} \circ \sigma_{y_0}) = \sigma_{z_1} \circ \sigma_{z_2} \circ \ldots \circ (\sigma_{y_1} \circ \sigma_{y_2} \circ \ldots)^{-1} \tag{6}$$

Given that $\sigma_{z_0} \circ \sigma_{y_0}$ represents the transpositions of the first layer and $(\sigma_{y_1} \circ \sigma_{y_2} \circ \ldots)^{-1}$ is the inverse of $(\sigma_{y_1} \circ \sigma_{y_2} \circ \ldots)$ (as the inverse of a transposition is itself), we find the following.

$$\sigma_{z'0} = \sigma_{z_1} \circ \sigma_{z_2} \circ \ldots \circ (\sigma_{y_1} \circ \sigma_{y_2} \circ \ldots)^{-1} = \sigma_{z_1} \circ \sigma_{z_2} \circ \ldots \circ \sigma_{y_2} \circ \sigma_{y_1} \tag{7}$$

Hence, $\sigma_{z'0} = \sigma_{z_1} \circ \sigma_{z_2} \circ \ldots \sigma_{y_2} \circ \sigma_{y_1}$. This equation has exactly the same situation as that described in Fact 3.2, $\sigma_{z'0}$ has to be the identity permutation, indicating $\sigma_{z_0} = \sigma_{y_0}$. This allows us to eliminate σ_{z_0} and σ_{y_0} from both sides of the equation. Repeating this process for every layer shows that all layer transpositions are identical. Therefore, $\sigma_{z_i} = \sigma_{y_i}$ for each layer, confirming that each includes the exact same set of disjoint transpositions.

Therefore, if the two permutations generated through FSS are ultimately the same, we can determine that all the transpositions included in between are identical. According to Fact 5.3, the minimum entropy and the Shannon entropy are identical in this case, and the maximum probability value for the existence of a single permutation is $\frac{1}{((n-1)\times N/2)}$, with all being equal. As FSS generates uniformly distributed permutations with possible configurations $((n-1) \times N/2)$, the minimum entropy of FSS is precisely $((n-1) \times N/2)$. □

Corollary 4.2. The total entropy of R-FSS is the number of random bits used.

Proof. By the Proposition 4.1, all permutation is distinguished up to using the same random bits. Thus, R-FSS, which is generated by uniformly distributed reduced random numbers, has the same entropy to the entropy of random bits. □

4.2 First order complexity of FSS

Theorem 4.3. If a random variable X follows the output distribution of $\mathsf{FSS}(R, x_i)$, where R consists of uniformly distributed random bits, then $\Gamma_1^{x_i}$ of any input x_i within the domain $[N]$, where $|[N]| = N = 2^n$, is $\frac{1}{N}$. Simply, 1st order attack probability of $\mathsf{FSS}(R, \cdot)$ is $\frac{1}{N}$.

Proof. Consider an arbitrary input $x \in [N]$ and denote y as the output produced by $y = \mathsf{FSS}(R, x)$, where R is a sequence of uniformly distributed random bits. To determine the probability that y equals $\mathsf{FSS}(R, x)$ for a randomly chosen x and y, we analyze the process layer by layer. In the initial layer, the input x is allocated to one of two possible groups with a probability of $\frac{1}{2}$. This binary selection process is repeated identically for each subsequent layer, up to the $\log(N)$-th layer. Therefore, the overall probability of obtaining a particular output y from input x under FSS is the cumulative product of the probabilities at each layer, calculated as $(\frac{1}{2})^{\log(N)} = \frac{1}{N}$. This calculation confirms that the likelihood of any specific outcome given any input is uniform $\frac{1}{N}$ across all possible outcomes. □

Based on Figure 7, the theorem states that there is only one unique way to connect a specific input node to a specific output node, which relies on the fact that the change in values (propagation direction) in each layer occurs with an equal probability of 1/2. Furthermore, according to Algorithm 2, the bit flip at each step can be randomly determined, and regardless of the initial input value, the output can be any value, with the probabilities intuitively being equal. Theorem 4.3 leads to the first-order attack is N. It can be used for the perfect security countermeasure against the 1st-order differential power analysis assumption. The 1st-order attack assumption is the most widely considered against side-channel attack.

4.3 q-th order analysis of FSS

As the next step, we aim to verify the security from the second order up to the q-th order. To carry out these complex calculations, we first want to address Fact 4.4.

Fact 4.4. *The q-th order attack probability* for $X = \binom{[N]}{q}$ is bounded below

$$\max([\pi_X]) \leq q! \times \max\left(\Gamma_1^{x_1} \times \Gamma_2^{x_2} \times \Gamma_3^{x_3} \times \ldots \times \Gamma_q^{x_q}\right) \tag{8}$$

for all subsets with q possible distinguished elements $x_1, x_2, \ldots, x_q \in \binom{[N]}{q}$.

Fact 4.4 is intuitively true, since $\Gamma_1^{x_1} \times \Gamma_2^{x_2} \times \Gamma_3^{x_3} \times \ldots \times \Gamma_q^{x_q}$ typically shows a maximum probability of a shuffled result with q inputs and q outputs. There are $q!$ cases for all combinations of q inputs. Thus, $\max([\pi_X])$ is bounded by the product of the number of all combinations $q!$ and the maximum value of the multiplication result of each probability.

Definition 4.5. A *Half-Recursive Sequence* is defined by the recursive formula below:

$$S_N = \left(\frac{1}{2}S_{N-1}\right),\ S_{N-1} \tag{9}$$

For example, if $S_0 = \{1, 2\}$, then $S_1 = \left\{\frac{1}{2}, 1, 1, 2\right\}$ and $S_2 = \left\{\frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1, \frac{1}{2}, 1, 1, 2\right\}$. In this paper, the important sequence that we use is initialized with $S_0 = \{1\}$. Consider its general form:

$$\begin{aligned}
S_0 &= \{1\},\\
S_1 &= \left\{\frac{1}{2}, 1\right\},\\
S_2 &= \left\{\frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1\right\},\\
S_3 &= \left\{\frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1\right\},\\
&\ldots\\
S_\gamma &= \left\{\frac{1}{2^\gamma}, \frac{1}{2^{\gamma-1}}, \frac{1}{2^{\gamma-1}}, \frac{1}{2^{\gamma-2}}, \ldots\right\}.
\end{aligned}$$

We observe that this sequence exhibits a fractal structure where $|S_\gamma| = 2^\gamma$. The overview of the sequence reflects the exact same structure as a small portion of the sequences. Assuming $S_\gamma = \{s_0, s_1, s_2, \ldots, s_{2^\gamma-1}\}$, we find $s_1 = s_2 = s_4 = s_8 = \ldots = s_{2^{\gamma-1}} = \frac{1}{2^{\gamma-1}}$. Looking at the subset of S_3 with all even-number indices, $\{s_0, s_2, s_4, s_6\} = \left\{\frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}\right\}$, which exactly matches the first four sequences of S_3. The main theorem addresses the worst-case probability $\pi(N)$.

Theorem 4.6. Given a Half-Recursive Sequence $S_{\log(N)} = \{s_0, s_1, \ldots, s_{N-1}\}$ initialized with $S_0 = \{1\}$, the q-th order attack probability for FSS on a set X chosen from $[N]$ with q elements, denoted by $X = \binom{[N]}{q}$, is bounded below

$$\max([\pi_X]) \leq q! \times s_0 \times s_1 \times \ldots \times s_{q-1}, \tag{10}$$

for any $q \leq N$.

Proof. By induction, we start the butterfly structure 2 indices ($\gamma = 1$). By Fact 4.4, it is trivial when $q = 1$, if $q = 2$, $S_1 = \{\frac{1}{2}, 1\}$ then,

$$\max\left(\Gamma_1^{x_1} \times \Gamma_2^{x_2} \times 2!\right) = s_0 \times s_1 \times 2! = \frac{1}{2} \times 1 \times 2! = 1. \tag{11}$$

There are only two indices; it follows the equation due to $\max([\pi_X]) = \binom{2}{2} = 1$ (see Figure 8).

Assume that it is true on k-depth ($\gamma = k$). Consider $(k+1)$ depth ($\gamma = k+1$). The butterfly network with $(k+1)$ depth is arranged in parallel by two k depth networks (see Figure 9), and each input index selects one of the two clusters. By assumption, the r-th order attack probability on $X = \binom{[N]}{r}$ ($r \leq 2^k$) of each of two k-depth clusters which has 2^k indices is

$$r! \times (\Gamma_1^{x_1(k)} \times \Gamma_2^{x_2(k)} \times \ldots \times \Gamma_r^{x_r(k)}) \leq r! \times (s_0^{(k)} \times s_1^{(k)} \times s_2^{(k)} \times \ldots \times s_{r-1}^{(k)}) \tag{12}$$

where, $s_j^k \in S_k$.

In other words, Equation 12 implies that it represents the maximum probability of the random variable $X = \binom{[N]}{r}$. Now, assume that the q-th order attack probability on the $(k+1)$-depth butterfly network ($q \leq 2^{k+1}$) is δ, and Equation 10 is incorrect. Then, there exists a set of inputs $\{x_1, x_2, \ldots, x_q\} \subseteq [N]$,

$$q! \times (\Gamma_1^{x_1(k+1)} \times \Gamma_2^{x_2(k+1)} \times \ldots \times \Gamma_q^{x_q(k+1)}) > q! \times (s_0^{(k+1)} \times s_1^{(k+1)} \times s_2^{(k+1)} \times \ldots \times s_{q-1}^{(k+1)}) \tag{13}$$

Because the factorial of the size of the index $q!$ has the same multiplication on both sides of the inequalities, we omit the number from now on. $(k+1)$ depth network is connected with two k depth networks with probability selection 1/2 in the first layer (see Figure 9). We can separate two parts of the depth network $(k+1)$, t and $(q-t) < t$, so let t be the number of inputs connected to the upper cluster k depth network, $(q-t)$ inputs connected to the other network ($t \leq N/2$). Thus, equation 13, can be described by k depth partially then $\frac{1}{2}\Gamma_i^{x_i(k)} = \Gamma_i^{x_i(k+1)}$ where $i \leq t$. Equation 13 is rewritten by,

$$\left(\frac{1}{2}\Gamma_1^{x_1(k)} \times \frac{1}{2}\Gamma_2^{x_2(k)} \times \ldots \times \Gamma_q^{x_q(k+1)}\right) > \left(\frac{1}{2}s_0^{(k)} \times \frac{1}{2}s_1^{(k)} \times \ldots \times s_{q-1}^{(k+1)}\right) \tag{14}$$

By assumption (Equation 12), left t elements can be removed then,

$$s_t^{(k+1)} \times s_{t+1}^{(k+1)} \times \ldots \times s_{q-1}^{(k+1)} < \Gamma_{t+1}^{x_{t+1}(k+1)} \times \Gamma_{t+2}^{x_{t+2}(k+1)} \times \ldots \times \Gamma_q^{x_q(k+1)} \tag{15}$$

On the other hand, the inputs form x_{t+1} to x_q goes to the second group of depth k, so $N/2 - t$ of elements of the left side of inequality (in Equation 15) is determined by a probability multiple with 1/2 or 1, so we know that below,

$$\Gamma_{t+1}^{x_{t+1}(k+1)} \times \Gamma_{t+2}^{x_{t+2}(k+1)} \times \ldots \times \Gamma_q^{x_q(k+1)} < \Gamma_1^{x_{t+1}(k)} \times \Gamma_2^{x_{t+2}(k)} \times \ldots \times \Gamma_{q-t-1}^{x_q(k)} \tag{16}$$

and by the basic assumption (Equation 12),

$$\Gamma_1^{x_{t+1}(k)} \times \Gamma_2^{x_{t+2}(k)} \times \ldots \times \Gamma_{q-t}^{x_q(k)} < (s_0^{(k)} \times s_1^{(k)} \times s_2^{(k)} \times \ldots \times s_{q-t-1}^{(k)}) \tag{17}$$

By equation 15, 16, 17, we get

$$s_t^{(k+1)} \times s_{t+1}^{(k+1)} \times \ldots \times s_{q-1}^{(k+1)} < (s_0^{(k)} \times s_1^{(k)} \times s_2^{(k)} \times \ldots \times s_{q-t-1}^{(k)}) \tag{18}$$

By the definition of a half-recursive sequence,

$$(s_0^{(k)} \times s_1^{(k)} \times s_2^{(k)} \times \ldots \times s_{q-t-1}^{(k)}) = (2s_0^{(k+1)} \times 2s_1^{(k+1)} \times 2s_2^{(k+1)} \times \ldots \times 2s_{q-t-1}^{(k+1)}) \tag{19}$$

Thus, we get the equation below.

$$s_t^{(k+1)} \times s_{t+1}^{(k+1)} \times \ldots \times s_{q-1}^{(k+1)} < (2s_0^{(k+1)} \times 2s_1^{(k+1)} \times 2s_2^{(k+1)} \times \ldots \times 2s_{q-t-1}^{(k+1)}) \tag{20}$$

Since $q - t < t$, it is in contradiction to Lemma B.4. Therefore, the assumption, (Equation 13), is rejected. Thus, $(k+1)$ depth butterfly network (FSS) holds Equation 10.

□

Fig. 8. Second order probability of 1-depth network (initially, two indices are opened to be two outputs, once one of input selected to be output by the probability 1/2, then the other thing is just determined)

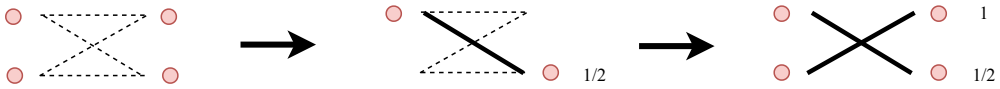
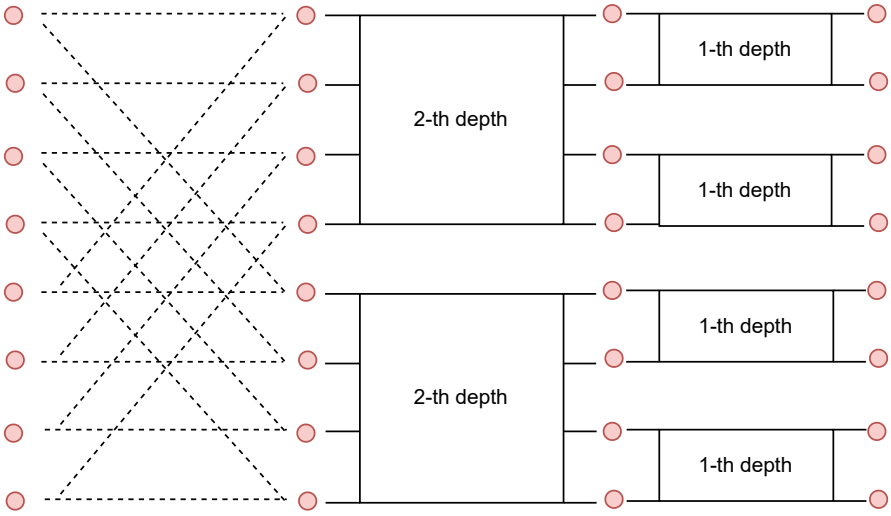


Fig. 9. 3-depth butterfly network connected to 1 and 2 depth networks) - Induction step



For example, if a 3rd-order attack is performed on 16 indices, by Theorem 4.6, the specific steps for calculating the attack complexity are as follows:

- A half recursive sequence with 16 elements is generated: $\left\{\frac{1}{16}, \frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{1}{1}\right\}$.
- Compute the bound below as $3! \times s_0 \times s_1 \times s_2 = 3! \times \frac{1}{16} \times \frac{1}{8} \times \frac{1}{8} \approx \frac{1}{171}$. Therefore, the 3-rd order attack probability is approximately $\frac{1}{171}$.
- The 3-rd order attack complexity is approximately $171 > 2^7$.

In the uniform random permutation having $N!$ cases, the 3rd-order attack complexity will be $\binom{16}{3} = 560 > 2^9$. Therefore, we can directly compare the complexity between the uniform random permutation and FSS. We will discuss comparisons of various applications with permutations in Section 5.

5 COMPARISON BETWEEN FSS AND q-WISE INDEPENDENT OF FULL DEPTH BUTTERFLY NETWORK (BENES NETWORK)

Algorithm 4 2× Fixed-Swap Shuffle; Benes Network - Permutation $\mathsf{BEN}(R,\cdot) = \mathsf{FSS}^{-1}(R,\cdot) \circ \mathsf{FSS}(R,\cdot)$ with deterministic random bits with length $(N) \times (\log(N) - 1)$

Input: An integer $x \in [N]$ such that $N = 2^n$ and random bit stream $R = r_0 r_1 r_2 \ldots r_{(N)\times(\log(N)-1)}$
Output: $\mathsf{BEN}(R, x) \in [N]$

for $j = 0$ to $n - 1$ **do**
  $s \leftarrow$ left rotation j bits of x
  $t \leftarrow$ right $(n-1)$ bits of x
  $t \leftarrow R_{t+(N/2)\times j}$ /*On the fly PRF with Random Bits*/
  if t equals 0 **then**
    $x \leftarrow x$ /*No operation*/
  else
    $x \leftarrow x \oplus 2^{(n-1)-j}$ /*1 bit Exclusive-OR*/
  end if
end for
for $j = (n-2)$ to 0 **do**
  $s \leftarrow$ left rotation j bits of x
  $t \leftarrow$ right $(n-1)$ bits of x
  $t \leftarrow R_{t+(N/2)\times(j+n)}$ /*On the fly PRF with Random Bits*/
  if t equals 0 **then**
    $x \leftarrow x$ /*No operation*/
  else
    $x \leftarrow x \oplus 2^{(n-1)-j}$ /*1 bit Exclusive-OR*/
  end if
end for
return x

5.1 q-set-wise independent and q-th order attack complexity

To determine the attack complexity of a full depth Benes Network, we decide to utilize an existing study [15]. The reasons for employing this method are as follows:

- This approach is convertible into our q-th order complexity. We show the exact way to compute the minimum entropy and Shannon entropy from TVD in this section.
- To our knowledge, this method has calculated the highest complexity for the cryptographic application of the Benes network.

Definition 5.1. (Almost q-set-wise Independence [15]). Let $\mathcal{D}$ be a distribution over $\mathbb{S}_n$, and q an integer. We say $\mathcal{D}$ is ϵ-almost q-set-wise independent in L^1-norm, if for any initial distribution v_0 over $X = \binom{[n]}{q}$,

$$\|D_X v_0 - \mathcal{U}_X\| \leq \epsilon$$

where D_X is the $|X| \times |X|$ Transition matrix and $\mathcal{U}_X$ is uniform distribution over X.

As we can see, intuitively, it has the exactly the same probabilities of matching of inputs and outputs in q-th order attack complexities. Based on E.Gelman et al. [15], Fact 5.2 is assumed to be true. This theorem directly tells us the TVD whit q-quarries.

Fact 5.2. For every q and n such that $\binom{q}{2} \leq n$, the Benes network is $\frac{q(q-1)}{2n}$-almost-q-set-wise independent.

To demonstrate the value of our q-th-order complexity evaluation, we will transform Fact 5.2 using entropy and worst-case probability, converting it into minimum entropy and Shannon entropy.

5.2 Deriving Lower-Bound Entropy Formula from TVD

Fact 5.3. Let $\mathcal{D}$ be a distribution on set X, the maximum entropy for distribution $\mathcal{D}$ is uniform. Also, the distribution of the minimum entropy is characterized by the conditions,

- One extremely biased case with high probability.
- Uniform distributed low probabilities for the others.

Proposition 5.4. (General formula for TVD to Entropy) Let $\mathcal{D}$ be a distribution over a random variable X on sample space S where $|S| = \omega$, and $\mathcal{U}$ is the uniform distribution on S is the set of all event of The total variation distance between $\mathcal{D}$ and $\mathcal{U}$ is ϵ, then the lower-bound Shannon entropy of D is

$$H(X) \geq \left(\frac{1}{\omega} + \epsilon\right) \log\left(\frac{1}{\omega} + \epsilon\right) + (\omega - 1)\frac{\frac{1}{\omega}(\omega - 2) - 2\epsilon + 1}{2(\omega - 1)} \log\left(\frac{\frac{1}{\omega}(\omega - 2) - 2\epsilon + 1}{2(\omega - 1)}\right)$$

The lower-bound minimum entropy of D is

$$H_{min}(X) = \frac{1}{\frac{1}{\omega} + \epsilon}$$

Proof. Let total variation distance between $\mathcal{D}$ and $\mathcal{U}$ be ϵ denote,

$$\frac{1}{2} \sum_{x \in X} |\mathcal{D} - \mathcal{U}| = \epsilon \tag{21}$$

Let $|X|$ = n then this equation is described as,

$$|p_1 - (1/\omega)| + |p_2 - (1/\omega)| + \ldots + |p_\omega - (1/\omega)| = 2\epsilon \tag{22}$$

The distribution with Total Variation Distance for lowest entropy is under conditions Fact 5.3. Therefore, $|P_1| > (|P_i|)$ and $|P_i| = |P_j|$ where $i, j \geq 2$. Also, we can use the sum of probability is 1 and Equation 22 is simplified by,

$$(p_1 - (1/\omega)) + (\omega - 1)((1/\omega) - p_2) = 2\epsilon \tag{23}$$

and,

$$p_1 + (\omega - 1)p_2 = 1 \tag{24}$$

p_1, p_2 are the computed as below, (detail calculation process is described in Appendix A)

$$p_1 = \frac{1}{\omega} + \epsilon \tag{25}$$

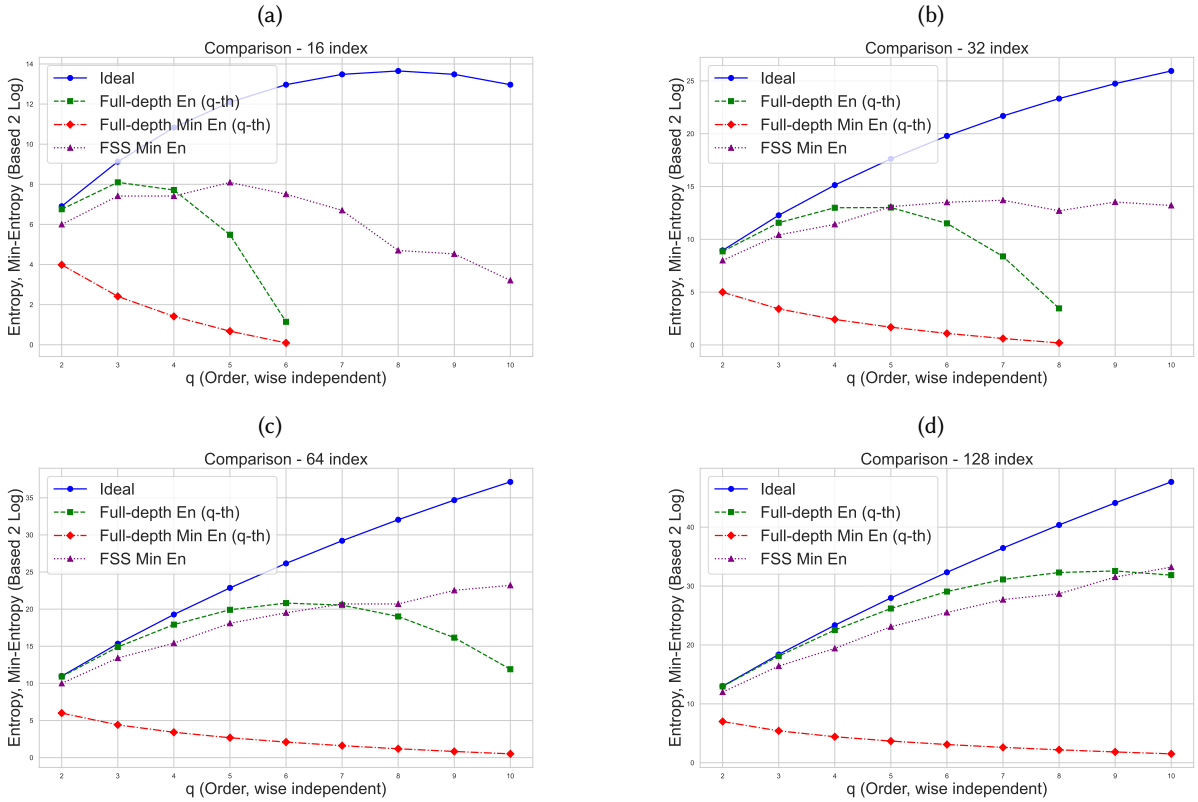


Fig. 10. x-axis : q with respect to each line, y-axis : Guessing Complexity (Entropy) with log scale. Complexity comparison among, ideal entropy with $\binom{N}{q}$ cases (Blue), minimum entropy (Red) and Shannon entropy (Green) derived from q-set-wise independent with Benes Network, and the minimum entropy of FSS computed by our method (Purple)

$$p_2 = \frac{\frac{1}{\omega}(\omega-2) - 2\epsilon + 1}{2(\omega-1)} \tag{26}$$

Finally, we can calculate the lower bound guessing entropy by $H(X) = -p_1 \log(p_1) + (n-1)p_2 \log(p_2)$,

$$H(X) \geq \left(\frac{1}{\omega} + \epsilon\right) \log\left(\frac{1}{\omega} + \epsilon\right) + (\omega - 1)\frac{\frac{1}{\omega}(\omega-2) - 2\epsilon + 1}{2(\omega-1)} \log\left(\frac{\frac{1}{\omega}(\omega-2) - 2\epsilon + 1}{2(\omega-1)}\right)$$

□

Since the minimum entropy is the same to q-th order complexity, we can directly use this value to the comparison to q-th order complexity of FSS.

5.3 Comparison with q-th order probability for FSS and q-wise independent for Benes Network

Since we only are interested in the aspect of higher-order side channel attack, we would like to know the q-th order attack complexity which is the worst-case entropy of q-quarries. Because Total Variation Distance (TVD) can be converted into Shannon entropy (Equation 25) and Min entropy (Equation 26), we were also able to directly compare with other papers that calculated TVD in butterfly networks or Thorp shuffling. In evaluating the security level of our q-th order according to our calculation method, we also made additional comparisons with Thorp shuffling by B.Morris et al. [30] and swap-or-not shuffling [18]. However, we were unable to obtain significant figures in scenarios with small numbers of rounds such as $\log(N)$ or $2\log(N)$, small domains ($N = 2^4$ to $N = 2^7$), and a small number of queries (essentially because the complexity was close to zero, making the comparison largely meaningless). Furthermore, we did not conduct a direct experimental comparison with Park et al. [35]. This is because that paper only details the complexity for the second order, and the results presented are purely experimental, with no theoretical formulation of the complexity for the q-th order described. Therefore, we would like to remind you that in this section we have only conducted comparisons with E.Gelman et al. [15] where $\binom{q}{2} \geq N$.

Figure 10 shows comparisons among the worst case entropy of FSS that is computed by Equation 10 in Theorem 4.6, the worst case entropy (computed by Equation 25) and Shannon entropy (computed by Equation 26) in Benes Network with $(2\log(N) - 1)$-depth in $\binom{q}{2} \geq N$, and the Ideal entropy assumed by perfect permutation which has $\log\binom{N}{q}$ entropy in $N = 16$, $N = 32$, $N = 64$, and $N = 128$ indices. Each index means certain applications; such as AES (16 sboxes [50]), ML-KEM (32 bytes of message [32]), and ML-DSA (128 vectors operated independently in NTT [33]).

We have shown that the results for the first-order complexity (1-set-wise independent) are all equal to Ideal Entropy. This fact has already been addressed in Theorem 4.3, and this part is omitted from the figure. The overall trend that can be seen from the figure is as follows. First, the worst-case entropy (Min-Entropy) calculated by q-set-wise independent is actually too small to provide a satisfactory result for the figures we want to compare. Second, the Shannon entropy calculated by q-wise independence shows a similar trend to the q-th order attack complexity of FSS, but it reverses FSS as q increases. It should be noted that there may be an illusion due to the different calculation methods between the two, but, of course, the Benes Network actually has a higher complexity. Therefore, our method of calculating attack complexity can much more effectively capture high attack complexity. Third, up to the third-order attack, the complexity between FSS and Ideal Permutation does not show a significant difference. For this part, it needs to be checked whether the difference with the ideal permutation is meaningful depending on the specific noise and attack situation.

6 LIMITATIONS, FUTURE WORKS AND CONCLUSION

Through this paper, we have shown the following results:

- We showed the butterfly network as an actual implementation algorithm.
- We showed the attack complexity from the perspective of countermeasures against side-channel analysis when only $\log(N)$ depth of butterfly network was applied.
- To prove the attack complexity, we defined a recursive structure and mathematically proved the actual attack complexity.
- We compared the attack complexity with the Benes Network, which was calculated in the existing method, and this does not mean that the actual attack complexity is higher, but rather that our calculation method is more suitable for showing the worse case probability.

Through this study, we have shown the potential of the butterfly network as an actual side channel attack countermeasure and expect to maximize performance in generating permutation sequences when parallel implementation is performed on actual hardware ASIC or FPGA. In particular, when implementing cryptographic

algorithms with large vector sizes, such as PQC algorithms, through software/hardware co-design, significant benefits can be obtained.

6.1 Limitations

Our results have the following limitations. First, our structure does not have a significant advantage over Fisher-Yates shuffling because it can only be computed sequentially when implemented in software. Depending on the implementation method, to our knowledge, we cannot implement FSS more efficiently with the most optimized FY shuffling using only multiplication and shift [34]. To get the most out of our structure, we need to parallelize it using hardware. When parallelized using hardware, FSS which is $\log(N)$-depth bit swaps are required, so it is possible to perform the operation in just one clock cycle, creating an extremely efficient algorithm. However, even in this case, there is a disadvantage that more random bits are required compared to the existing results [35], and it can be seen that this point has been emphasized in the existing results. Of course, a structure can be integrated with a pseudo-random function that takes limited seed input, such as SHAKE256 [31]. However, even in this case, various external conditions, such as the timing of performing SHAKE operations, must be considered additionally.

6.2 future works

In addition, if we assume that encryption operations are performed using FSS, our result (Theorems 4.3 and 4.6) does not provide enough evidence to determine whether this encryption method is secure. The q-th order complexity is the worst case, so when computing the boundary needed to calculate the actual Total Variation Distance, a value that is excessively large compared to the actual TVD result is calculated. However, we believe that $\log(N)$ depth or two or three connections of FSS will be sufficient to use as a secure block cipher, and more research is needed. In this case, it is much more advantageous in terms of efficiency in terms of computational methods than the existing swap-or-not approach [18].

Therefore, we will conduct the following research in the future:

- Research on ways to reduce random bits while ensuring q-th order security
- Research on ways to prove security as a general block cipher.

ACKNOWLEDGMENTS

Thanks God.

REFERENCES

[1] 2022. NIST round 4 submission. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. Accessed on: Sep 2022.

[2] Axel Bacher, Olivier Bodini, Alexandros Hollender, and Jérémie Lumbroso. 2015. Mergeshuffle: A very fast, parallel random permutation algorithm. *arXiv preprint arXiv:1508.03167* (2015).

[3] Ali Galip Bayrak, Nikola Velickovic, Paolo Ienne, and Wayne Burleson. 2012. An architecture-independent instruction shuffler to protect against side-channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 1–19.

[4] V. Benes. 1964. Optimal Rearrangeable Multistage Connecting Networks. *Bell System Technical Journal* (1964).

[5] Patrick Billingsley. 2017. *Probability and measure*. John Wiley & Sons.

[6] John Black and Phillip Rogaway. 2002. Ciphers with arbitrary finite domains. In *Topics in Cryptology - CT-RSA 2002: The Cryptographers' Track at the RSA Conference 2002 San Jose, CA, USA, February 18–22, 2002 Proceedings*. Springer, 114–130.

[7] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop*. Springer, 16–29.

[8] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. 2003. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop*. Springer, 13–28.

[9] Zhaohui Chen, Yuan Ma, and Jiwu Jing. 2022. 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* 42, 1 (2022), 322–326.

[10] Guojing Cong and David A Bader. 2005. An Empirical Analysis of Parallel Random Permutation Algorithms ON SMPs.. In *PDCS*. 27–34.
[11] Artur Czumaj. 2015. Random permutations using switching networks. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. 703–712.
[12] Morris Dworkin. 2016. Recommendation for block cipher modes of operation. *NIST special publication* 800 (2016), 38G.
[13] Ronald Aylmer Fisher and Frank Yates. 1953. *Statistical tables for biological, agricultural, and medical research*. Hafner Publishing Company.
[14] P Gallagher. 2009. Federal information processing standards publication digital signature standard (DSS). *Fips pub 186-3* (2009).
[15] Efraim Gelman and Amnon Ta-Shma. 2014. The Benes Network is q*(q-1)/2n-Almost q-set-wise Independent. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
[16] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. 2022. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), 223–263.
[17] Jens Gustedt. 2008. Engineering parallel in-place random generation of integer permutations. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 129–141.
[18] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. 2012. An enciphering scheme based on a card shuffle. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Springer, 1–13.
[19] Donald E Knuth. 2014. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional.
[20] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Advances in Cryptology–CRYPTO'99: 19th Annual International Cryptology Conference*. Springer, 388–397.
[21] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology–CRYPTO'96: 16th Annual International Cryptology Conference*. Springer, 104–113.
[22] Daniel Langr, Pavel Tvrdík, Tomáš Dytrych, and Jerry P Draayer. 2014. Algorithm 947: Paraperm—Parallel Generation of Random Permutations with MPI. *ACM Transactions on Mathematical Software (TOMS)* 41, 1 (2014), 1–26.
[23] Stefan Mangard. 2003. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Information Security and Cryptology–ICISC 2002: 5th International Conference*. Springer, 343–358.
[24] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2008. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media.
[25] Thomas S Messerges. 2000. Using second-order power analysis to attack DPA resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 238–251.
[26] Rory Mitchell, Daniel Stokes, Eibe Frank, and Geoffrey Holmes. 2022. Bandwidth-optimal random shuffling for GPUs. *ACM Transactions on Parallel Computing* 9, 1 (2022), 1–20.
[27] Ben Morris. 2008. The Mixing Time of the Thorp Shuffle. *SIAM J. Comput.* 38, 2 (2008), 484–504. https://doi.org/10.1137/050636231 arXiv:https://doi.org/10.1137/050636231
[28] Ben Morris. 2009. Improved mixing time bounds for the Thorp shuffle and L-reversal chain. *The Annals of Probability* 37, 2 (2009), 453 – 477. https://doi.org/10.1214/08-AOP409
[29] Ben Morris. 2013. Improved mixing time bounds for the Thorp shuffle. *Combinatorics, Probability and Computing* 22, 1 (2013), 118–132.
[30] Ben Morris, Phillip Rogaway, and Till Stegers. 2018. Deterministic encryption with the Thorp shuffle. *Journal of Cryptology* 31 (2018), 521–536.
[31] NIST. 2015. SHA-3 standardization. https://csrc.nist.gov/Projects/Hash-Functions/SHA-3-Project/SHA-3-Standardization.
[32] NIST. 2022. *FIPS 203 "Module-Lattice-Based Key-Encapsulation Mechanism Standard"*. Technical Report. Accessed on: Sep 2022.
[33] NIST. 2022. *FIPS 204 "Module-Lattice-Based Digital Signature Standard"*. Technical Report. Accessed on: Sep 2022.
[34] Jong-Yeon Park, Jang-Won Ju, Wonil Lee, Bo Gyeong Kang, Yasuyuki Kachi, and Kouichi Sakurai. 2024. A statistical verification method of random permutations for hiding countermeasure against side-channel attacks. *Journal of Information Security and Applications* 84 (2024), 103797. https://doi.org/10.1016/j.jisa.2024.103797
[35] Jong-Yeon Park, Dongsoo Lee, Seonggyeom Kim, Wonil lee, Bo Gyeong Kang, and Kouichi Sakurai. 2023. Fully Parallel, One-Cycle Random Shuffling for Efficient Countermeasure in Post-Quantum Cryptography. Cryptology ePrint Archive, Paper 2023/1889. https://eprint.iacr.org/2023/1889 https://eprint.iacr.org/2023/1889.
[36] Manuel Penschuck. 2023. Engineering Shared-Memory Parallel Shuffling to Generate Random Permutations In-Place. *arXiv preprint arXiv:2302.03317* (2023).
[37] Peter Pessl. 2016. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India*. Springer, 153–170.
[38] Peter Pessl and Robert Primas. 2019. 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, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6*. Springer, 130–149.

[39] Robert Primas, Peter Pessl, and Stefan Mangard. 2017. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 513–533.
[40] Zehua Qiao, Yuejun Liu, Yongbin Zhou, Yuhan Zhao, and Shuyi Chen. 2024. Single Trace is All It Takes: Efficient Side-channel Attack on Dilithium. *Cryptology ePrint Archive* (2024).
[41] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay. 2020. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT. In *Security, Privacy, and Applied Cryptography Engineering (Lecture Notes in Computer Science, Vol. 12586)*, L. Batina, S. Picek, and M. Mondal (Eds.). Springer, Cham. https://doi.org/10.1007/978-3-030-66626-2_7
[42] Thomas Ristenpart and Scott Yilek. 2013. The mix-and-cut shuffle: small-domain encryption secure against N queries. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. Springer, 392–409.
[43] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.
[44] John K Salmon, Mark A Moraes, Ron O Dror, and David E Shaw. 2011. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*. 1–12.
[45] Ahmad Shabani and Bijan Alizadeh. 2020. Enhancing hardware trojan detection sensitivity using partition-based shuffling scheme. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 1 (2020), 266–270.
[46] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.
[47] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. IEEE, 124–134.
[48] Julian Shun, Yan Gu, Guy E Blelloch, Jeremy T Fineman, and Phillip B Gibbons. 2014. 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, 431–448.
[49] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. 2020. Single-trace attacks on message encoding in lattice-based KEMs. *IEEE Access* 8 (2020), 183175–183191.
[50] NIST-FIPS Standard. 2001. Announcing the advanced encryption standard (AES). *Federal Information Processing Standards Publication* 197, 1-51 (2001), 3–3.
[51] Andrew S Tanenbaum. 2003. *Computer networks*. Pearson Education India.
[52] Stefan Tillich, Christoph Herbst, and Stefan Mangard. 2007. Protecting AES software implementations on 32-bit processors against power analysis. In *Applied Cryptography and Network Security: 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007. Proceedings*. Springer Berlin Heidelberg, 141–157.
[53] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. 2012. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. Springer, 740–757.
[54] Jason Waddle and David Wagner. 2004. Towards efficient second-order power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 1–15.
[55] Yoo-Seung Won, Bo-Yeon Sim, and Jong-Yeon Park. 2020. Key schedule against template attack-based simple power analysis on a single target. *Applied Sciences* 10, 11 (2020), 3804.
[56] Timo Zijlstra, Karim Bigou, and Arnaud Tisserand. 2019. FPGA implementation and comparison of protections against SCAs for RLWE. In *Progress in Cryptology–INDOCRYPT 2019: 20th International Conference on Cryptology in India, Hyderabad, India, December 15–18, 2019, Proceedings 20*. Springer, 535–555.

A COMPUTATION OF EQUATION

This section show the calculation process for Equation 25 and 26. We start the equations below,

$$(p_1 - \frac{1}{\omega}) + (\omega - 1)(\frac{1}{\omega} - p_2) = 2\epsilon$$

and,

$$p_1 + (\omega - 1)p_2 = 1$$

Let $q = 1/\omega$ then,

$$(p_1 - q) + (\omega - 1)(q - p_2) = 2\epsilon$$

We can remove p_1 in the equation with $p_1 = 1 - (\omega - 1)p_2$. as below,

$$(1 - (\omega - 1)p_2 - q) + (\omega - 1)(q - p_2) =$$
$$-2(\omega - 1)p_2 + 1 - q + (\omega - 1)q = 2\epsilon$$

Therefore,

$$-2(\omega - 1)p_2 = 2\epsilon - 1 - \omega q + 2q$$

Thus,

$$p_2 = \frac{q(\omega - 2) - 2\epsilon + 1}{2(\omega - 1)}$$

We can also simply get,

$$p_1 = 1 - \frac{q(\omega - 2) - 2\epsilon + 1}{2} = \frac{1}{\omega} + \epsilon$$

B PROOF OF LEMMAS

Definition B.1. A set $S = \{s_0, s_1, s_2, \ldots, s_\tau\}$ is ordered set. Let $\Omega \subseteq S$, $\Omega = \{s_\eta, \ldots, s_\kappa\}$ is *left-most-subset* of S if $\eta = 0$, *right-most-subset* of S if $\kappa = \tau$.

Definition B.2. $[\alpha, \beta]$*-subset of ordered set* S is a ordered subset $\{s_\alpha, s_{\alpha+1}, \ldots, s_{\beta-1}\}$ such that $\beta - \alpha = 2^k$ for a $k \in \mathbb{N}$ and α is multiple of 2^k.

Fig. 11. An example for $[\alpha, \beta]$-Subset; [16,24]-Subset, [20,24]-Subset, and [20,22]-Subset

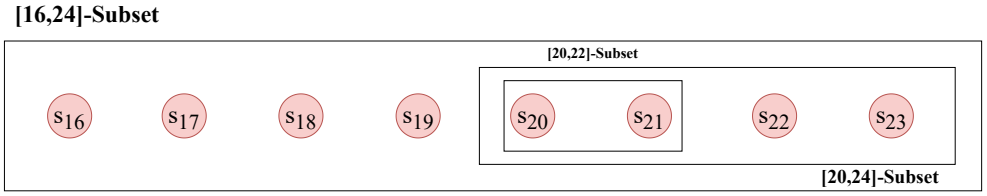


Figure 11 shows examples of $[\alpha, \beta]$-subset, such as $[16, 24]$-subset, $[20, 24]$-subset, and $[20, 22]$-subset. All of them have the properties as $\beta - \alpha = 2^k$ and α is a multiple of 2^k, $16 = 2^3 \times 2$, $20 = 2^2 \times 5$ and $20 = 2^1 \times 10$.

Lemma B.3. Let S be a half recursive sequence and Ω_j, Δ_j be subset of S. $\Omega_j, \Delta_j \subseteq [\alpha, \beta]$-subset with $\gamma = \frac{(\beta-\alpha)}{2}$,$|\Omega_j| = |\Delta_j|$, $\Omega_j = \{s_{\omega_j}, s_{\omega_j+1}, \ldots s_\sigma\}$, $\Delta_j = \{s_\tau \ldots s_{\delta_j-1}, s_{\delta_j}\}$, and at least one of each set is left-most-subset of $[\alpha, \beta]$-subset or right-most-subset of $[\alpha, \beta]$-subset, such that $s_{\omega_j} = s_\alpha$ or $s_{\delta_j} = s_{\beta-1}$. Let Δ_{j+1} and Ω_{j+1} be defined as below,

$$\Omega_{j+1} = \{s_i \in \Omega_j | s_{\gamma+i} \notin \Delta_j\}$$
$$\Delta_{j+1} = \{s_i \in \Delta_j | s_{i-\gamma} \notin \Omega_j\}$$

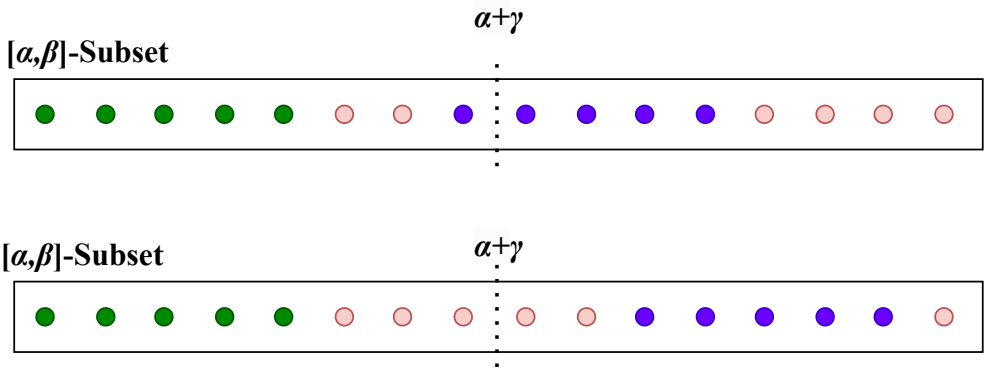
(We call this procedure *Folding*), Then, there exist $\iota \geq \alpha, \kappa \leq \beta$, one of Ω_{j+1} or Δ_{j+1} is left-most-subset or right-most-subset of $[\iota, \kappa]$-subset.

Proof. Consider $\Omega'_j = \{s_{\gamma+\omega_j}, s_{\gamma+\omega_j+1}, \ldots s_{\gamma+\sigma}\}$. If $\Omega'_j \cap \Delta_j = \phi$, then $\Omega_j = \Omega_{j+1}$ and $\Delta_j = \Delta_{j+1}$. Thus, we only need to consider $\Omega'_j \cap \Delta_j \neq \phi$. There are two cases according to the definition of Ω_j and Δ_j.
1) Ω_j is left-most-subset of $[\alpha, \beta]$-subset.
2) Δ_j is right-most-subset of $[\alpha, \beta]$-subset.

The first case and second case have the symmetric structure, we need to show just the first case, and the other case is naturally concluded by the symmetric method. If $s_{\gamma+\omega_j}$ is not included in Δ_j, then Ω_{j+1} has the element s_α. Thus, Ω_{j+1} is right-most-subset of $[\alpha, \beta]$-subset, see the lower case of Figure 12. If $s_{\gamma+\omega_j}$ is included in Δ_j, then the elements of high indices in Ω_j and Δ_j are not in Ω_{j+1} and Δ_{j+1}, which have the indices $s_{\omega_j+\gamma}$ to s_{δ_j-1}. Thus, the highest index element of $\Delta_{j+1} = s_{\gamma+\omega_j-1}$. γ is a power of 2. Δ_{j+1} is right-most-subset of $[\alpha, \alpha+\gamma]$-subset, see the upper case of Figure 12. The second case is also true symmetrically.

□

Fig. 12. Tow case for the location of $\xi = \alpha + \gamma$. Upper case shows ξ cuts S_δ, lower case does not cut S_δ. (Where S_Ω is the left-most-subset of $[\alpha, \beta]$-Subset ($\beta = \alpha + 16$, $\gamma = 8 = \frac{16}{2}$), and the relation between S_Ω and S_δ where $|S_\Omega| = |S_\delta| = 5$).



Lemma B.4. Let S be a half-recursive sequence, $S = \{s_0, s_1, s_2, \ldots, s_{N-1}\}$ and two subsets composed by consecutive $\eta \leq N$, $\Omega_1 = \{s_0, s_1, \ldots, s_{\eta-1}\}$, $\Delta_1 = \{s_\omega, s_{\omega+1}, \ldots, s_{\omega+\eta-1}\}$ where $\omega \geq \eta$. *Then,*

$$\prod_{s_i \in \Omega_1} (2 \times (s_i)) \leq \prod_{s_i \in \Delta_1} s_i \tag{27}$$

Proof. In Lemma B.3, the addition or subtraction $\gamma = \frac{\beta-\alpha}{2}$ in $[\alpha, \beta]$-subset means that adding γ doubles the element's value, while subtracting γ halves its value. There exists a smallest $k \in \mathbb{N}$ such that Ω_1 is the most-left-subset of $[0, 2^k]$-subset of S. We can produce Ω_j and Δ_j, and this step reduces the number of elements of subsets by Lemma B.3. Considering that as each step progresses, some elements are removed, and the removed elements have a difference of twice, if all elements disappear at a certain stage, the proof is complete.

Assume that there are some elements in Ω_i and Δ_i in the smallest $[\alpha_i, \beta_i]$-subset when there is no further progress possible. Since adding $\gamma_i = \frac{\beta_i-\alpha_i}{2}$ to all elements of Ω_i does not meet any elements of the Δ_i, create a

new set by subtracting γ from all elements of Δ_i. This means that all elements of Ω_i are doubled in this step. The proof is complete.

□