Lightweight Encryption using Chaffing and Winnowing with All-or-Nothing Transform for Network-on-Chip Architectures

Hansika Weerasena, Subodha Charles and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Network-on-Chip (NoC) fulfills the communication requirements of modern System-on-Chip (SoC) architectures. Due to the resource-constrained nature of NoC-based SoCs, it is a major challenge to secure on-chip communication against eavesdropping attacks using traditional encryption methods. In this paper, we propose a lightweight encryption technique using chaffing and winnowing (C&W) with all-or-nothing transform (AONT) that benefits from the unique NoC traffic characteristics. Our experimental results demonstrate that our proposed encryption technique provides the required security with significantly less area and energy overhead compared to the state-of-the-art approaches.**

***Index Terms*—system-on-chip, network-on-chip, security**

I. Introduction

The advancement of manufacturing technologies has enabled the integration of more and more diverse intellectual property (IP) cores on the same system-on-chip (SoC). Commercial SoCs, such as the Intel "Xeon Phi" series [1] and Tilera "TILE-Gx" family [2], consist of SoCs up to 72 cores. The demand for scalable and high-throughput on-chip interconnects has made network-on-chip (NoC) the standard interconnection solution for many-core SoCs [2]. While optimizing the SoC for performance and energy efficiency is a primary objective, recent manufacturing trends have raised several security concerns [3]. Due to cost as well as time constraints, manufacturers outsource IPs to third-party vendors across the globe [4]. Typically, a few important IPs are manufactured in-house and are integrated with third-party IPs to obtain the final SoC. As a result of this distributed supply chain, malicious implants such as hardware Trojans can be inserted into the IPs. Trojans can be inserted into the RTL or into the netlist of an IP core with the intention of launching attacks without being detected at the post-silicon verification stage or during runtime [5]. There are several practical scenarios of Trojan insertion during the long and distributed supply chain such as by an untrusted CAD tool or designer or at the foundry via reverse engineering. Given the importance of trustworthy computing, there are many research efforts in efficient detection and mitigation of security vulnerabilities [4], [6]–[14].

The major security issues related to NoC can be classified as eavesdropping, spoofing, denial-of-service, buffer overflow, and side-channel attacks [9]. Among the many IP cores integrated on the SoC, some of them will have secure, crucial information. Since NoC facilities communication between all the IPs, it exposes an ideal threat vector for an attacker to exploit. This allows the attacker to eavesdrop on the NoC packets to extract secret information without hacking into individual IPs. Therefore, protecting the packets transferred on an NoC is a major concern. However, the added security must not cause significant performance and energy efficiency degradation. Complex security schemes that counter the extraction of secret information such as AES encryption [15] can have a significant impact on overall SoC performance and power consumption, and as a result, they are not suitable for the resource-constrained NoC-based SoCs.

Authenticated encryption is a widely used solution against eavesdropping attacks. Figure 1 shows a typical NoC implementation on a many-core SoC where packets are encrypted when transferring between IPs. Previous work on securing on-chip communication proposed several lightweight encryption schemes [8], [16]–[19]. However, all these solutions took the traditional path of encryption using block ciphers and made it lightweight by using techniques such as reducing the number of rounds, smaller key sizes, etc. This leads to sub-optimal

This work was partially supported by NSF grant SaTC-1936040.

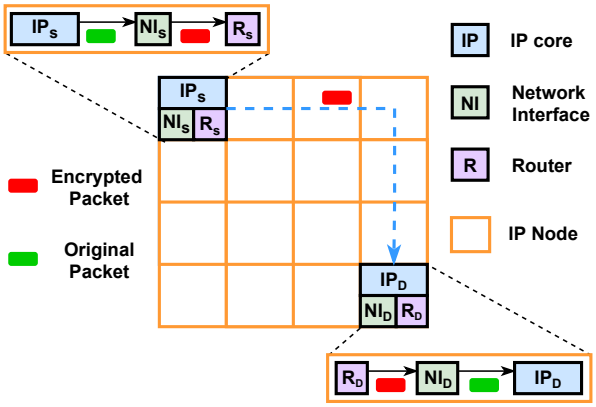

Fig. 1: NoC implementation using a 4x4 mesh topology. Each IP is connected to NoC via a network interface and a router. To avoid eavesdropping attack, communication from source (IP_S) to destination (IP_D) is encrypted.

results in terms of performance, power consumption, and security.

This paper tries to answer a fundamental question: ***is it possible to develop a novel encryption scheme that utilizes the unique characteristics of NoC traffic to derive a lightweight solution while providing the desired security?*** We assume a strong threat model where IPs and several routers can be malicious. The network interfaces that interface the NoC IP to other IPs are typically manufactured in-house and are assumed to be secure. A similar threat model has been the focus of several other studies, which validates the reality of the model [6], [8], [16], [17], [19], [20].

Research Contributions: Our work introduces a lightweight key-less encryption scheme using chaffing and winnowing (C&W) together with an all-or-nothing transform (AONT) that utilizes the unique NoC traffic characteristics. Specifically, this paper makes the following important contributions.

- We propose to replace the traditional block cipher-based encryption using a key-less encryption scheme that utilizes C&W and AONT.
- The performance overhead of deriving realistic "chaff" packets to be dispersed among the "wheat" packets is addressed using NoC traffic characteristics that allocate a predictable sequence number to every packet injected from the same IP.
- We show that the combination of C&W and AONT can provide the desired security guarantees for NoCs.
- Experimental results demonstrate that our approach is an ideal fit for resource-constrained SoCs since it incurs significantly lower performance, area and energy overhead compared to traditional encryption.

The rest of the paper is organized as follows. Section II presents background information and prior research efforts. Section III describes our approach for lightweight encryption. Section IV presents the experimental results. Finally, Section V concludes the paper.

II. Background and Related Work

A. All or Nothing Transform

All or Nothing Transform (AONT) is a concept originally introduced by Rivest in 1997 [21] to increase the difficulty of launching a brute force attack on encryption algorithms. An AONT is not considered encryption. It is an invertible, unkeyed, randomized transformation, which acts as a pre-processing step prior to encryption. The main property of an AONT is that when a message is transformed using AONT and then encrypted using a block cipher-based encryption mode, an adversary cannot reveal any information about the message (not even one block) without decrypting all the blocks. For example, typically, to decrypt a block encrypted with a k-bit key using a brute force approach requires 2^k work in the worst case or 2^{k-1} on average. In the ECB (electronic codebook) mode, the adversary only needs to decrypt the first ciphertext block to obtain the corresponding plaintext block. However, using an AONT as a pre-processing step before ECB can increase the work required by orders of magnitude, depending on the encrypted message size. This is helpful in scenarios where the key space size is fixed and the encryption algorithm is considered to be "marginal" [22], such as the 56-bit DES [23].

AONT maps the message sequence $(m_1, m_2, ..., m_n)$ to $(m'_1, m'_2, ..., m'_n)$ such that,

1) The transformation is invertible.
2) AONT and its inverse are effectively computable.
3) It is infeasible to recover the whole message if at least w bits of the transformation are unknown (encrypted), where w is a threshold related to the security guarantees and in most cases, the size of an AONT block.

There are several AONT implementations including package transform [21], optimal asymmetric encryption padding [24], a variant of the package transform based on the counter mode of encryption [25], exposure-resilient function-based transform [26] and quasigroup-based transform [27], [28]. In this paper, we use an adaptation of the AONT proposed in [28] due to its applicability in resource-constrained environments.

B. Quasigroups and Latin Squares

A finite quasigroup $\langle Q, \bullet \rangle$ of order n consists of a set Q in which a binary operator $\bullet$ is defined that has the following property: $\forall a, b \in Q$, the equations $a \bullet x = b$ and $y \bullet a = b$, always have unique solutions for x and y. A dual binary operator $\circ$ can be defined for the binary operator $\bullet$ with the following relationship using the elements of set Q:

$$a \circ b = c \iff a \bullet c = b$$

A new finite quasigroup $\langle Q, \circ \rangle$ can be derived which is the dual of $\langle Q, \bullet \rangle$. Using the dual operation, we can derive the relationship:

$$a \circ (a \bullet b) = b; a \bullet (a \circ b) = b$$

A Latin Square (LS) is an $n \times n$ matrix defined on n symbols such that every row and every column contain exactly one occurrence of a specific symbol. For $n = 3$, there are 12 possible LSs and for $n = 4$, there are 576, which shows that possible LSs grow exponentially with n (sequence A002860 in OEIS [29]).

The multiplication table of a quasigroup of size n is a LS of the same order [30]. Therefore, construction of LSs provides ways of generating random quasigroups. Marnes introduced a fast and randomized method to generate a quasigroup from a LS [27]. Their approach required the order of the LS to be $n = p - 1$, where p is a prime number. This method can produce $n!$ (n factorial) distinct random LSs of order n. Even though using this method reduces the number of possible LSs, it is obvious that the LS space is still considerably large to prevent brute force attacks (for $n = 256$, LS space $\approx 8.5 \times 10^{506}$). Since the primary objective of this paper is to develop a lightweight encryption scheme, we use an adaptation of this approach for quasigroup generation (detailed in Section III-B).

C. Chaffing and Winnowing

Chaffing and winnowing (C&W) is a technique that offers confidentiality without encryption [31]. The technique which is named after its similarities with removing chaff from wheat, consists of two main processes completed at the two ends of communication - sender and receiver (source IP and destination IP in our case). At the sender, a message authentication code (MAC) is appended to each packet, which is computed using a standard hash-based MAC algorithm [32]. Therefore, each packet originating at the source IP has the packet payload as plaintext and the MAC. Typically, a message consists of multiple packets and therefore, each packet consists of a sequence number to determine the order of the packets that combine to create the message. The sequence number also helps remove duplicate packets and identify missing packets. For example, a message can have the following packet sequence represented with the format *(sequence number, payload, MAC)*:

(1, Our next meeting, 230985)
(2, will be at the docks, 992405)
(3, at midnight June 28, 128476)

These are called "wheat" packets. However, if this is sent as it is, the message is not encrypted and any eavesdropper can read the packet data. Therefore, in C&W, confidentially is achieved by adding "chaff" packets to the communication stream. Chaff packets are fake packets that have the same format and similar-looking content and are intermingled with the wheat packets. A value from a uniformly random distribution is used as the MAC replacement and therefore, the MAC of each chaff packet is invalid. For example, the above packet sequence after adding chaff can look as follows:

(1, Our next meeting, 230985)
(1, Our next call, 366357)
(2, will be at the docks, 992405)
(2, will be on Signal, 098121)
(3, at midnight June 28, 128476)
(3, at noon May 18, 471298)

The receiver has to discard the chaff packets and retain only the wheat packets to construct the correct message. To do this, the receiver validates the MAC of each packet and discards the packets with invalid MACs. It is important to note that this process of discarding packets with invalid MACs takes place anyway in a typical packet-based communication system that implements authentication. An eavesdropper who can not validate the MAC is unable to "winnow out" the chaff and therefore, unable to retrieve the correct message.

The security of a C&W scheme depends on the number of C&W packets, the security of the MAC algorithm and the way chaffing is done. Bellare and Boldyreva [33] critically evaluated the security of C&W on different notions of security. Their work showed that a *bit-by-bit* C&W scheme provides "find-then-guess" security. However, the bit-by-bit scheme requires adding chaff for every wheat bit of the packet, which drastically increases overhead. Therefore, we use an adaptation of the bit-by-bit C&W scheme (detailed in Section III-C) in our encryption scheme that fits the low overhead requirements of NoC-based SoCs while providing provable security.

D. Fermat Primes

Fermat primes are prime numbers that can be expressed in the form of $2^{2^n} + 1$ where n is a non-negative

integer. Currently, there are only five Fermat primes that have been discovered (sequence A019434 in OEIS [29]). The known Fermat primes are; $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, and $F_4 = 65537$. Our approach of quasigroup generation (introduced in Section II-B and elaborated in detail in Section III-C) and other steps in AONT (Section III-A) motivates the usage of Fermat primes.

E. Related Work

Eavesdropping attacks in NoC have been addressed by authenticated encryption in prior research efforts [16], [17], [34]. Content of the packets originating from source IP is encrypted (C) except for the header information (H) and the message authentication tag (T). The packet injected to the NoC consists of $H \parallel C \parallel T$. Having the header information as plain text helps the routers to process the packets faster and send to the relevant destination. At the destination, the tag is validated to check for tampering and the packet is decrypted.

Sepúlveda et al. [16] proposed a variation of authenticated encryption as a tunnel-based communication mechanism where only the destination headers are kept as plain text. The authors used AES counter mode for encryption [15] and Siphash [35] for authentication on source headers and data. Although AES counter mode is highly parallelizable for performance gain, it introduces high area and energy overhead. Siphash is a lightweight and fast hash function well suited for short inputs and is an ideal fit for NoC-based SoCs.

Previous work tried to develop lightweight encryption schemes to fit the resource-constrained nature of NoC. A simple XoR cipher together with a packet certification technique was proposed in [19]. Intel introduced TinyCrypt - a cryptographic library targeting resource-constrained IoT and embedded devices that provides basic functionalities with less overhead. Boraten et al. [36] proposed a reconfigurable packet validation and authentication scheme by merging two robust error detection schemes, namely, algebraic manipulation detection and cyclic redundancy check. However, these approaches either lead to unacceptable design overhead or do not provide the required security guarantees.

AONT has been used in a wide variety of applications in previous studies. AONT is used as a countermeasure for differential power analysis based side-channel attack in [37]. AONT is also used to protect implementations of cryptographic algorithms against partial key exposures in exposure resilient cryptography [38]. To the best of our knowledge, our proposed approach is the first attempt in using AONT with C&W while leveraging the unique NoC traffic characteristics to secure packet transfers in NoC architectures.

TABLE I: Table of notations.

p	One time define Fermat prime
n	Size of quasigroup where $n = p - 1$
s	no of AONT blocks
$\sigma(u)$	A function that returns a random permutation of elements $1, ..., u$ of size u.
K'	Key used to derive the first row of the LS. $\left\|K'\right\| = n$
M	Message to be encrypted
M'	Pseudo-Message after AONT
C	Cipher Text
B_i	Block in AONT where $\|B_i\| = n$
d_i	$d_i \in (1, 2, ..., n)$ and $\|d_i\| = \log_2 n$ bits
$\langle Q, \bullet \rangle$	LS defined by binary operator $\bullet$ of a quasigroup
q_{ij}	Element in row i and column j in LS
w	no of C&W bits
$\bar{b_i}$	inverse of bit b_i
$A * B$	$\forall\ a_i \in A\ \&\ b_i \in B$; $(a_i \times b_i) \bmod p$
$\{0,1\}^k$	A random permutation of k bits
$MAC(K, in)$	MAC using key K: $\{0,1\}^k \rightarrow \{0,1\}^z$
$dt[i]$	i^{th} bit of dt bit stream
$[\,]$	empty string

III. Encryption with C&W and AONT

Figure 2 presents an overview of our approach lightweight encryption framework. The encryption and decryption hardware will be implemented in the network interface since the network interfaces are assumed to be secure according to the threat model. The remainder of this section describes our proposed encryption and decryption procedure using the notations outlined in Table I.

Algorithm 1 Encryption

1: **function** $E_K(M)$
2: $M' \leftarrow AONT(M)$
3: parse M' as $m' \parallel m''$ where $\left|m'\right| = w$ bits
4: $c' \leftarrow e_K(m')$
5: $C \leftarrow c' \parallel m''$
6: **return** C

Algorithm 1 shows the major steps of the proposed encryption procedure. As the initial setup step, global values of p and w are selected depending on the security requirements. The selection of p and w is described in detail in Section IV. When the message (M) is sent from the source IP (IP_S), it is first transformed using *AONT* which outputs the pseudo message (M') (line 2). Then, M' is divided as m' and m'' depending on the value of w (line 3). The first part (m'), which is

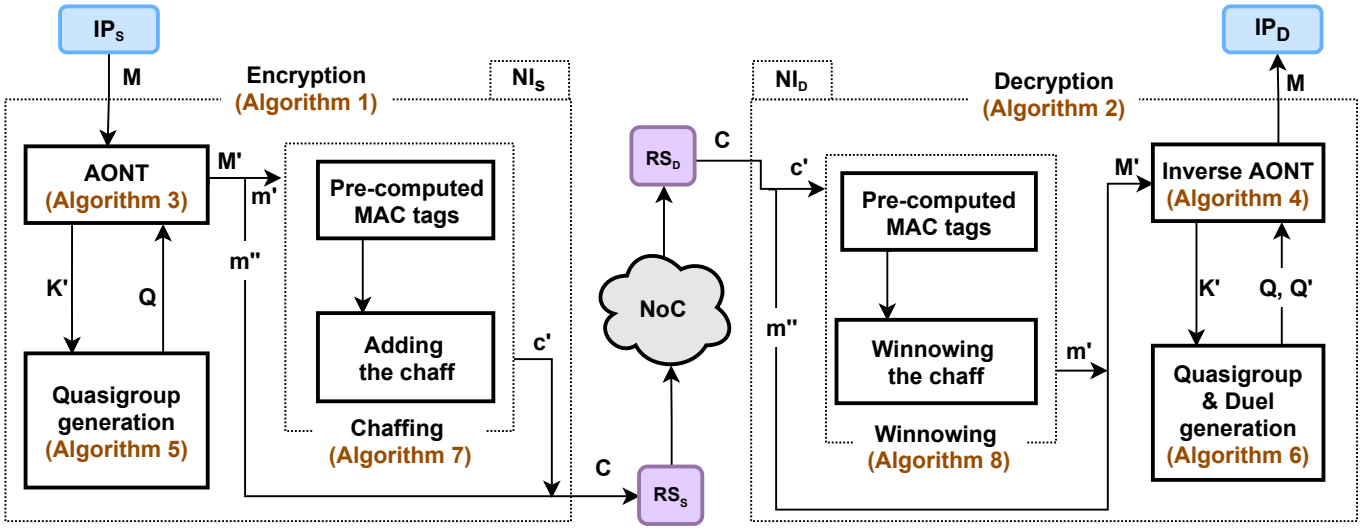

Fig. 2: Overview of our proposed lightweight encryption framework with chaffing and winnowing (C&W) with all-or-nothing-transform (AONT). The packet sent from the source (IP_S) goes through AONT followed by chaffing which is implemented in the source network interface (NI_S). The generated ciphertext (C) traverses the NoC to the destination. The destination network interface (NI_D) decrypts the packet using winnowing followed by inverse AONT (IAONT) and delivers it to the destination IP (IP_D).

w-bits long, undergoes bit-by-bit C&W to produce the "chaffed" output c' (line 4). Finally, c' is concatenated with m'' to form the final ciphertext C (line 5).

Algorithm 2 Decryption

1: **function** $E_K(C)$
2: parse C as $c' \parallel m''$
3: $m' \leftarrow d_K(c')$
4: $M' \leftarrow m' \parallel m''$
5: $M \leftarrow IAONT(M')$
6: **return** M

Decryption follows the reverse of encryption as outlined in Algorithm 2. First, the message (C) is divided into c' and m'' (line 2). Next, c' is sent to the "winnowing" process which discards the bits with invalid MACs and returns m' (line 3). Finally, m' is concatenated with the m'' to form the M' (line 4) before applying inverse *AONT* to produce the original message M (line 5). The required MACs for both wheat and chaff bits can be pre-computed since the sequence number of the packets originating at an IP are predictable (incremented by one for each packet).

The remainder of this section elaborates all these components in detail. Section III-A explains the functions $AONT$ and $IAONT$. Section III-B outlines the generation of LSs, which are used in the AONT. Section III-C elaborates the chaffing (e_K) and winnowing (d_K) processes.

A. All or Nothing Transformation (AONT)

Algorithm 3 describes the AONT invoked in line 2 of Algorithm 1. The message is first transformed to do the arithmetic operations in base n (line 2). For the base transformation to be computationally less demanding, we chose n to be a power of two, which motivated us to use a Fermat prime for p. To achieve this, the message is divided into groups of bits of length $\log_2 n$. Then, each group is represented by a number in base n where its symbols are mapped to the integers $\{1, ..., n\}$ (for example when $n = 4$, $(00)_2$ can be represented as 4 and $(11)_2$ as 3). The resulting string is then divided into s blocks of size n. K' is generated as a random permutation of $\{1, ..., n\}$ (line 3) which is then sent as input to Algorithm 5 in Section III-B to generate LS ($\langle Q, \bullet \rangle$)(line 4).

The *leader* (l) which is required as an initial value is generated through element-wise application of binary operator using the $\langle Q, \bullet \rangle$ (line 5). A single iteration of a for loop from line 6 to 9 maps message block (B_i) to a pseudo message block (B'_i):

- Line 7: integer i is represented using a number in base n of length n (for example $i = 1 = (0001)_4$ can be represented as (4,4,4,1).
- Line 8: intermediate block $E(i)$ is calculated by applying the binary operator on the elements of $I(i)$. The previously calculated leader is used to calculate n^{th} element of $E(i)$.
- Line 9: Pseudo message block (B'_i) is generated

Algorithm 3 All or Nothing Transform

1: **function** AONT(M)
2: parse M as $B_1 \parallel B_2 \parallel ... \parallel B_s$ where $B_i = (d_{i1}, d_{i2}, ..., d_{in})$
3: $K' \leftarrow \sigma(n)$
4: $\langle Q, \bullet \rangle \leftarrow Q(K')$
5: $l_1 \leftarrow k_1, l_2 \leftarrow k_2 \bullet l_1, ..., l_n \leftarrow k_n \bullet l_{n-1}$ and $l \leftarrow l_n$
6: **for** $i = 1, ..., s$ **do**
7: $I(i) \leftarrow$ representation of i to number in base n and $|I(i)| = n$
8: $E(i) \leftarrow (e_{i1}, e_{i2}, ..., e_{in})$ where $e_{in} \leftarrow l \bullet i_{in}, e_{in-1} \leftarrow e_{in} \bullet i_{in-1}, ..., e_{i1} \leftarrow e_{i2} \bullet i_{i1}$
9: $B'_i \leftarrow (d'_{i1}, d'_{i2}, ..., d'_{in})$ where $d'_{ij} \leftarrow e_{ij} \bullet d_{ij}, j = 1, ..., n$
10: $A_1 \leftarrow B'_1$
11: **for** i=2,...,s **do**
12: $A_i \leftarrow A_{i-1} * B_i$
13: $B'_{s+1} \leftarrow A_s * K'$
14: $M' \leftarrow B'_i \parallel B'_2 \parallel ... \parallel B'_{s+1}$
15: **return** M'

from the corresponding message block (B_i) by applying the binary operator $\bullet$ element by element with the corresponding $E(i)$ calculated in line 10.

Lines 10 to 13 generate the last pseudo block (B'_{s+1}). Auxiliary blocks (A_i) are calculated by applying the star operator between A_{i-1} and B_i (line 12). Finally, the $s+1$ concatenated (line 14) blocks are returned as the pseudo message (M').

Algorithm 4 elaborates the function $IAONT$, which is the inverse of $AONT$ with the following changes:

- Line 6: element by element division of A_s from B'_{S+1} is used to retrieve K' based on:
$$(a * k')mod\ p = m' \iff a/m' = k'$$
- Line 7: both LSs including the dual is generated from Algorithm 6.
- Line 12: dual operator $\circ$ is used to generate the original message blocks from pseudo message blocks.

B. Quasigroups and Latin Squares Generation

Our approach uses the LS generation technique introduced by Marnes et al. [27] since it leads to a fast and randomized generation process (Algorithm 5). The size of the LS is $n \times n$ where $n = p - 1$ and p is a prime. The first row of the LS which is a random permutation is provided as a parameter to the algorithm (line 2). Every element of the i-th row, $i = 2, ..., n$ is computed by

Algorithm 4 Inverse All or Nothing Transform

1: **function** $IAONT(M')$
2: parse M' as $B'_1 \parallel B'_2 \parallel ... \parallel B'_s$ where $B'_i = (d'_{i1}, d'_{i2}, ..., d'_{in})$
3: $A_1 \leftarrow B'_1$
4: **for** i=2,...,s **do**
5: $A_i \leftarrow A_{i-1} * B'_i$
6: $K' \leftarrow A_s / B'_{s+1}$
7: $\langle Q, \bullet \rangle, \langle Q, \circ \rangle \leftarrow Q'(K')$
8: $l_1 \leftarrow k_1, l_2 \leftarrow k_2 \bullet l_1, ..., l_n \leftarrow k_n \bullet l_{n-1}$ and $l \leftarrow l_n$
9: **for** $i = 1, ..., s$ **do**
10: $I(i) \leftarrow$ representation of i to number in base n and $|I(i)| = n$
11: $E(i) \leftarrow (e_{i1}, e_{i2}, ..., e_{in})$ where $e_{in} \leftarrow l \bullet i_{in}, e_{in-1} \leftarrow e_{in} \bullet i_{in-1}, ..., e_{i1} \leftarrow e_{i2} \bullet i_{i1}$
12: $B(i) \leftarrow (d_{i1}, d_{i2}, ..., d_{in})$ where $d_{ij} \leftarrow e_{ij} \circ d'_{ij}, j = 1, ..., n$
13: $M \leftarrow B_i \parallel B_2 \parallel ... \parallel B_{s+1}$
14: **return** M

Algorithm 5 Generate Quasigroup

1: **function** $Q(K')$
2: parse K' as $q_{11} \parallel q_{12} \parallel ... \parallel q_{1n}$ ▷ first raw of LS
3: **for** $i = 2, ..., n$ **do**
4: **for** $j = 1, ..., n$ **do**
5: $q_{ij} \leftarrow (i \times q_{1j}) \bmod p$
6: $\langle Q, \bullet \rangle \leftarrow$ LS of q_{ij}
7: **return** $\langle Q, \bullet \rangle$

$(i \times q_{1j}) \bmod p$ (line 5). The generated LS is then used in the AONT calculations.

Algorithm 6 is used by the receiver to generate both the original quasigroup and its dual simultaneously as LSs. For every element (q_{ij}) of row i and column j, a corresponding dual element (q'_{iz}) of value j is generated for row i and column $q_{ij}(z)$ (lines 6 and 7).

C. Chaffing and Winnowing

The input message (m') of length w is used in the bit by bit C&W process. The final output has $2w$ packets because there is a chaff bit for each bit in m'. Algorithm 7 outlines the process of chaffing. The m' is treated as a bit stream and the steps shown in line 3 to line 7 are applied on every bit of m'. A tag ($tg^{i,b}$) is generated by a secure MAC algorithm for the bit b_i and counter (ctr) using the shared authentication key K (line 4). In our implementation, we used NoC packet sequence

Algorithm 6 Generate Quasigroup with Dual

1: **function** $Q'(K')$
2: parse K' as $q_{11} \parallel q_{12} \parallel ... \parallel q_{1n}$
3: **for** $i = 1, ..., n$ **do**
4: **for** $j = 1, ..., n$ **do**
5: $q_{ij} \leftarrow (i \times q_{1j}) \bmod p$
6: $z \leftarrow q_{ij}$
7: $q'_{iz} \leftarrow j$
8: $\langle Q, \bullet \rangle \leftarrow$ LS of q_{ij}
9: $\langle Q, \circ \rangle \leftarrow$ LS of q'_{iz}
10: **return** $\langle Q, \bullet \rangle$, $\langle Q, \circ \rangle$

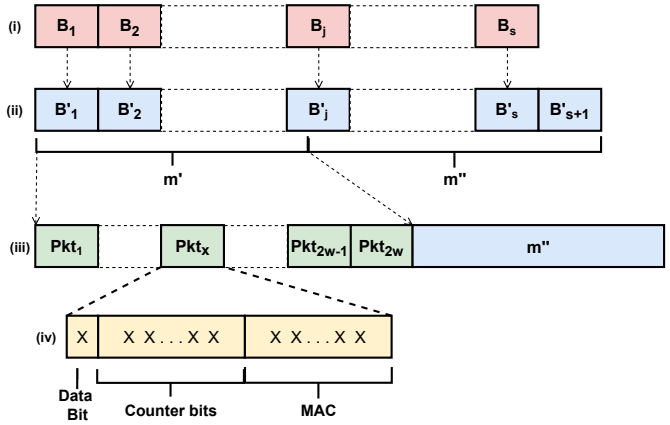

Fig. 3: Breakdown of bits from AONT and C&W: (i) original packet divided into blocks for AONT, (ii) transformed blocks after AONT, (iii) m' converted to $2w$ C&W packets and, (iv) overview of a chaffed packet.

number and an offset as the value of ctr. A random tag ($tg^{i,\bar{b}}$) is generated for the complement of the bit (line 5). Two packets are generated for the original bit and the complement bit as a combination of the bit, ctr, and tag (line 6 & 7).

In our approach, MAC can be generated in parallel, without waiting for the bits (m') to arrive since the sequence number of each packet is predictable in NoC architectures and only one bit is used from m' for each $tg^{i,b}$. This enables significant performance improvement.

Algorithm 7 Chaffing

1: **function** $e_K(m')$
2: break m' into bits as $b_1 \parallel b_2 \parallel ... \parallel b_w$
3: **for** $i = 1, ..., w$ **do**
4: $tg^{i,b} \leftarrow MAC(K, b_i \parallel \langle ctr + i \rangle)$
5: $tg^{i,\bar{b}} \xleftarrow{R} \{0,1\}^{|K|}$
6: $Pkt^{i,0} \leftarrow (b_i \parallel \langle ctr + i \rangle, tg^{i,0})$
7: $Pkt^{i,1} \leftarrow (\bar{b_i} \parallel \langle ctr + i \rangle, tg^{i,1})$
8: **return** $Pkt^{1,0}, Pkt^{1,1}, ..., Pkt^{w,0}, Pkt^{w,1}$

Algorithm 8 Winnowing

1: **function** $d_K(Pkt_1, ..., Pkt_{2w})$
2: $m' \leftarrow [\,]$
3: **for** $i = 1, ..., 2w$ **do**
4: parse Pkt_i as $dt_i \parallel tg_i$
5: **if** $MAC(K, d_1) = tg_i$ **then**
6: $m' \parallel dt_i[1]$
7: **return** m'

Algorithm 8 outlines the winnowing process. It takes $2w$ packets and outputs the original bit stream of length w. Each packet is validated using the same MAC algorithm and the key K (line 5). Invalid packets are discarded and the bit values of the valid packets are concatenated to produce the original message. The chaffing process increases the number of flits since for each bit in m', a tag and a counter are appended. However, as shown in experimental results, the impact on performance due to the increase in congestion is compensated by the faster encryption (and decryption). Figure 3 elaborates how the bits are composed in the final ciphertext compared to the original message.

D. An Illustrative Example

In this section, we provide an illustrative example to show how AONT and C&W work together to secure NoC packets. Let M be the message sent by the sender.

M = 0101 1111 0110 1000 1101 1010 1011 1100 1110 0001

p and w are set to 5 and 4, respectively. Since $p = 5$, $n = 4$ as illustrated in Section III-A. Let the chosen alphabet be 1,2,3,4 where the binary equivalent of 00 is represented by 4. Therefore, the message M can be represented as,

M = 11331224312223343241

where the blocks are,

$$B_1 = (1,1,3,3) \quad B_2 = (1,2,2,4)$$
$$B_3 = (3,1,2,2) \quad B_4 = (2,3,3,4)$$
$$B_5 = (3,2,4,1)$$

Assume that the derived random key is:

$$K' = (3,2,4,1)$$

Figure 4a shows the LS $\langle Q, \bullet \rangle$ constructed by Algorithm 5.

The leader can be calculated as,

$l_1 = 3, l_2 = 2 \bullet 3 = 3, l_3 = 4 \bullet 3 = 1, l_4 = 1 \bullet 1 = 3$ and $l = 3$

(a) $\langle Q, \bullet \rangle$

$\bullet$	1	2	3	4
1	3	2	4	1
2	1	4	3	2
3	4	1	2	3
4	2	3	1	4

(b) $\langle Q, \circ \rangle$

$\circ$	1	2	3	4
1	4	2	1	3
2	1	4	3	2
3	2	3	4	1
4	3	1	2	4

Fig. 4: Example of Latin Square of order 4 and its dual generated by Algorithm 5 and 6.

Calculation of $I(i)$:

$$I(1) = (4,4,4,1) \quad I(2) = (4,4,4,2)$$
$$I(3) = (4,4,4,3) \quad I(4) = (4,4,1,4)$$
$$I(5) = (4,4,1,1)$$

Calculation of $E(i)$:

$$E(1) = (4,4,4,4) \quad E(2) = (3,3,3,3)$$
$$E(3) = (2,2,2,2) \quad E(4) = (4,4,4,3)$$
$$E(5) = (2,2,2,4)$$

Calculation of pseudo-message blocks b'_i:

$$B'_1 = (2,2,1,1) \quad B'_2 = (4,1,1,3)$$
$$B'_3 = (3,1,4,4) \quad B'_4 = (3,1,1,3)$$
$$B'_5 = (3,4,2,2)$$

Calculation of Auxiliary blocks A_i:

$$A_1 = (2,2,1,1) \quad A_2 = (3,2,1,3)$$
$$A_3 = (4,2,4,2) \quad A_4 = (2,2,4,1)$$
$$A_5 = (1,3,3,2)$$

Using the above results, the final pseudo-message block can be calculated as,

$$B'_6 = A_5 * K = (1,3,3,2) * (3,2,4,1) = (3,1,2,2)$$

M'= 2211 4113 3144 3113 3422 ***3122***

The last block of M', which is ***3122*** in this example is the extra block added by AONT. Pseudo-message binary representation:

M'= 10100101 00010111 11010000 11010111 11001010 11011010

Let $w = 4$ and $ctr = 1$ for C&W. Then,

$m' = 1010$, $m'' =$ 0101 00010111 11010000 11010111 11001010 11011010

Let the outputs of of tg be,

$$tg^{1,1} = 1011 \quad tg^{2,0} = 1010$$
$$tg^{3,1} = 1110 \quad tg^{4,0} = 1111$$

Using the calculated tag values to derive and chaff and wheat packets, we get,

Wheat packets:

$$Pkt^{1,1} = (1,1,1011) \quad Pkt^{2,0} = (0,2,1010)$$
$$Pkt^{3,1} = (1,3,1110) \quad Pkt^{4,0} = (0,4,1111)$$

Chaff packets:

$$Pkt^{1,0} = (0,1,0011) \quad Pkt^{2,1} = (1,2,0011)$$
$$Pkt^{3,0} = (0,3,0111) \quad Pkt^{4,1} = (1,4,0101)$$

Then, the c' can be computed as,

$$c' = Pkt^{1,0} \parallel Pkt^{1,1} \parallel Pkt^{2,0} \parallel Pkt^{2,1} \parallel Pkt^{3,0} \parallel Pkt^{3,1} \parallel Pkt^{4,0} \parallel Pkt^{4,1}$$

If $|ctr| = 3$ bits, binary representation of c' is:

$$c' = 0\,001\,0011 \parallel 1\,001\,1011 \parallel 0\,010\,1010 \parallel 1\,010\,0011 \parallel 0\,011\,0111 \parallel 1\,011\,1110 \parallel 0\,100\,1111 \parallel 1\,100\,0101$$

Finally, ciphertext (C) is $c' \parallel m''$.

At the receiver's side, the winnowing process constructs m' by winnowing the chaff as outlined in Algorithm 8 and constructs M'. The $IAONT$ process parses M' to derives B'_i and A_i and retrieves the key as,

$$K = (1,3,3,2)/(3,1,2,2) = (3,2,4,1)$$

Both LSs $\langle Q, \bullet \rangle$ and its dual $\langle Q, \circ \rangle$ are constructed using Algorithm 6 as in Figure 4. Calculated $I(i)$ and $E(i)$ will be similar to the senders' side. Original blocks (B_i) of the message (M) can be constructed using B'_i and $\langle Q, \circ \rangle$ LS.

IV. Experiments

In this section, we first describe the experimental setup used to evaluate the effectiveness of our approach. Next, we present results to demonstrate the performance gain achieved using our approach compared to traditional symmetric key encryption. Finally, we discuss the associated overhead and security of our approach.

A. Experimental Setup

We used the cycle-accurate full system gem5 simulator to evaluate our approach [39]. The "GARNET2.0" model was used as on-chip interconnection model [40]. The configuration parameters used in gem5 is outlined in Table II.

We modified the Network Interface (NI) of gem5 source to simulate the proposed approach as well as the traditional encryption. Multiple benchmarks from SPLASH-2 and PARSEC benchmarks were run as applications to capture performance. To evaluate the area and energy overhead of our approach against traditional encryption, we synthesized both methods using Synopsys Design Compiler with "lsi_10k" library.

TABLE II: gem5 configuration parameters.

Processor configuration	
Number of cores	*16*
Core frequency	*2GHz*
Instruction Set Architecture	*x86*
Memory System Configuration	
L1 instruction cache	*private separate cache of 16kB*
L2 data cache	*private separate cache of 16kB*
Cache coherence	*directory-based cache coherence protocol*
Memory size	*4GB DDR*
Interconnection Network Configuration	
Topology	*4x4 Mesh topology*
Routing Scheme	*X-Y deterministic*
Link Latency	*1 Cycle*

C&W and AONT parameters: GARNET2.0 default implementation has data packet size of 576 bits. This motivated us to use a LS size(n) of 16 which led to an AONT block size of 64 bits. Both the counter and the MAC tag size are kept as 8 bits. The number of C&W bits (w) is kept as a variable which can be chosen according to the desired level of security.

Traditional Encryption: We compared our approach with symmetric encryption of AES-128 [41]. Since our AONT implementation works on blocks in parallel, we compared it with AES-128 in parallel CTR mode of encryption to enable a fair comparison. In this case, 576 bits of data require 5 parallel block ciphers of AES-128.

B. Performance Evaluation

The performance of our approach (**C&W and AONT**) is compared with two other scenarios: i) **No security** - NoC architecture that does not implement encryption to secure communication, and ii) **AES-128 parallel CTR** - packets secured using five AES-128 ciphers in counter mode. Figure 5 and Figure 6 show the average packet latency and overall execution time, respectively, with varying w values when running the FFT benchmark from the SPLASH-2 benchmark suite.

Packet latency is the number of cycles taken by one packet to traverse from source to destination. There is an average packet latency even in the "No security" scenario because of delays at the network interface, links, and routers in the NoC. AES-128 parallel CTR has high packet latency due to additional encryption operations taking place at the network interface. Overall execution time consists of CPU cycles, memory load/store delays in addition to the delays traversing the NoC.

Our proposed AONT implementation introduces $nlog_2n$ number of bits to packets which is constant for the selected LS. However, C&W introduces a variable

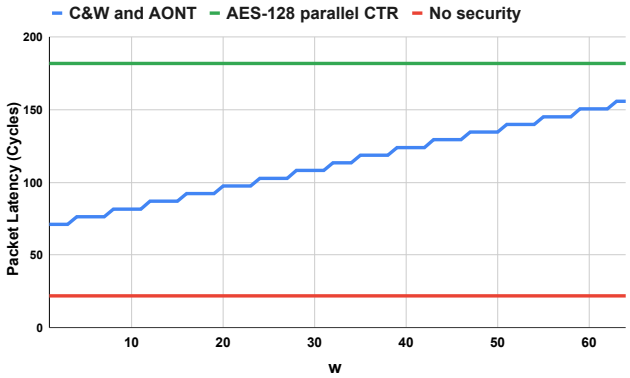

Fig. 5: Comparison of average packet latency with variable C&W bits against AES-128 and no security for FFT.

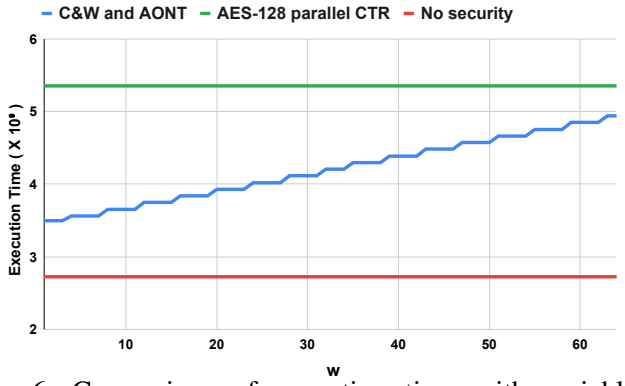

Fig. 6: Comparison of execution time with variable C&W bits against AES-128 and no security for FFT.

amount of bits ($2w(|ctr|+|tag|+1)-w$) depending on the number of C&W bits (w). The increasing number of bits contributes to the increasing number of flits injected into the network and as a result, increased packet latency. However, the performance penalty due to congestion is compensated by faster encryption in our approach.

We choose $w = 64$ experimentally according to the observations. We evaluated our approach against traditional encryption across multiple benchmarks of SPLASH-2 and PARSEC, namely, FFT, OCEAN, RADIX, LU, and Blackscholes. Figure 7 presents average packet latency and Figure 8 presents overall execution time across multiple benchmarks. It can be observed that our proposed scheme behaves similarly across all benchmarks. Our approach offers 14.3% improvement in packet latency and 7.7% improvement in overall execution time compared to traditional AES-128 encryption.

C. Overhead Analysis

Table III presents results based on the area and energy consumption calculations considering the same three scenarios. Each network interface must implement the required additional hardware for the security mechanism.

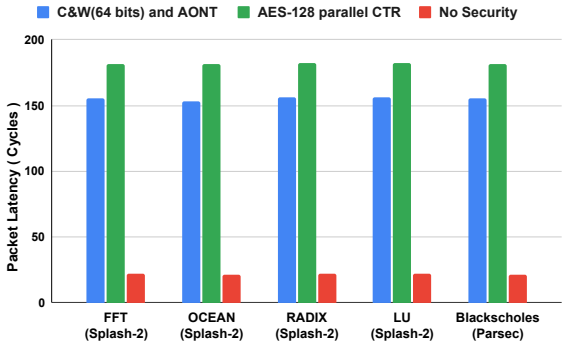

Fig. 7: Average packet latency comparison using traditional encryption, no security and C&W with AONT.

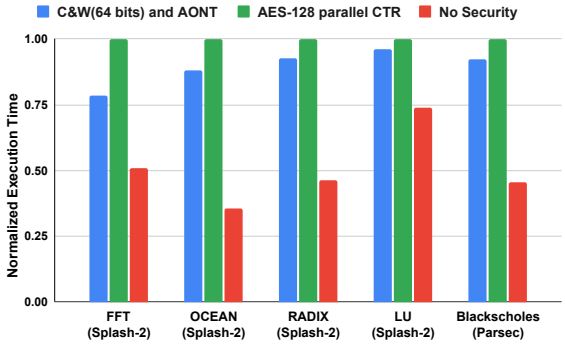

Fig. 8: Execution time comparison using traditional encryption, no security and C&W with AONT.

For the ease of illustration, we show the overhead at one network interface in Table III. Our approach improves the area overhead by 48.1% and energy efficiency by 72.1% compared to traditional encryption. The energy consumption is calculated for encrypting a 576-bit message. Our approach increases the energy efficiency significantly since AES-128 takes longer to encrypt and also, requires more power to run the five block ciphers in parallel. Therefore, our approach is ideal for resource-constrained NoC architectures.

D. Security Analysis

Security of our approach, which utilizes both bit-by-bit C&W and AONT, depends on the security of two main components: i) the implementation of the MAC algorithm used in the C&W scheme, and ii) security of the AONT scheme [33]. Bit-by-bit C&W scheme is proven to provide "find-then-guess" security assuming the underlying MAC is a pseudo-random function [33]. AONT scheme used in our approach is introduced as a secure AONT scheme in [28] as it followed the steps of package transform defined using quasigroup [21]. The package transform is proven to be secure with a strong semantic security model [24].

TABLE III: Comparison of area and energy overhead between traditional encryption and C&W with AONT.

	AES-128 parallel CTR	**C&W(64 bit) with AONT**	**Improvement**
Area	1505781	780164	48.1%
Energy(μJ)	16.6	4.6	72.1%

If fewer bits are used for C&W, the advantage of the adversary is higher. The advantage decreases exponentially with the increasing number of bits for C&W (w) because of the find-then-guess notion of security in bit-by-bit C&W scheme. This makes the security of our proposed approach configurable based on the security requirement and performance overhead.

Exhaustive key search attack is not possible in our approach compared to traditional encryption because AONT is key-less. For example, if $w = 64$, $s = 9$ and $n = 16$, an adversary needs to brute force $2^{(64+9)}/2$ trials on average, which is 2^{72} trials to recover the first row of the LS (K'). Also, it will be changed in the next message and there are 16! (n!) number of possible K' values. In other words, the attacker has to perform 2^{72} trials for every message, which is infeasible in practice.

The usage of AONT has also shown to hinder the possibility of differential side-channel analysis according to [37]. This can be considered as an added advantage of our approach over traditional encryption. Therefore, our approach is sufficiently secure in resource constraint environments such as NoC-based SoCs.

V. Conclusion

Network-on-Chip (NoC) is a widely used solution for on-chip communication between Intellectual Property (IP) modules in System-on-Chip (SoC) architectures. The increased usage of NoC and its distributed nature across the chip has made it a focal point of potential security attacks. It may not be feasible to implement costly encryption schemes on resource-constrained NoC-based SoCs. While parallel encryption methods can mitigate performance overhead, they can lead to unacceptable area and power penalties. In this paper, we proposed a lightweight encryption scheme based on chaffing and winnowing with all-or-nothing transform. Our chaffing and winnowing algorithms can be tuned to address the trade-off between security and design overhead. Experimental results demonstrated that our proposed approach can provide the desired security guarantees while incurring significantly lower energy and performance overhead compared to the state-of-the-art encryption methods.

REFERENCES

[1] G. Chrysos, "Intel® xeon phi™ coprocessor-the architecture," *Intel Whitepaper*, vol. 176, p. 43, 2014.

[2] C. Ramey, "Tile-gx100 manycore processor: Acceleration interfaces and architecture," in *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, 2011, pp. 1–21.

[3] P. Mishra and S. Charles, *Network-on-Chip Security and Privacy*. Springer Nature, 2021.

[4] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of dos attacks in noc based socs," in *Design Automation & Test in Europe (DATE)*, 2019, pp. 1160–1165.

[5] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP security and trust*. Springer, 2017.

[6] S. Charles, M. Logan, and P. Mishra, "Lightweight Anonymous Routing in NoC based SoCs," in *Design Automation & Test in Europe (DATE)*, 2020.

[7] S. Charles and P. Mishra, "Lightweight and trust-aware routing in noc-based socs," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 160–167.

[8] S. Charles and P. Mishra, "Reconfigurable network-on-chip security architecture," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 6, pp. 1–25, 2020.

[9] S. Charles and P. Mishra, "A survey of network-on-chip security attacks and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[10] H. Witharana, Y. Lyu, and P. Mishra, "Directed test generation for activation of security assertions in rtl models," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 4, pp. 1–28, 2021.

[11] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-Chip Security: Validation and Verification*. Springer Nature, 2019.

[12] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[13] A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, "Scalable hardware trojan activation by interleaving concrete simulation and symbolic execution," in *2018 IEEE International Test Conference (ITC)*. IEEE, 2018, pp. 1–10.

[14] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of distributed dos attacks in noc based socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[15] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[16] J. Sepúlveda, A. Zankl, D. Flórez, and G. Sigl, "Towards protected mpsoc communication for information protection against a malicious noc," *Procedia computer science*, vol. 108, pp. 1103–1112, 2017.

[17] S. Charles and P. Mishra, "Securing network-on-chip using incremental cryptography," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 168–175.

[18] "Using TinyCrypt Library, Intel Developer Zone, Intel, 2016." https://software.intel.com/en-us/node/734330, [Online].

[19] D. M. Ancajas *et al.*, "Fort-NOCs: Mitigating the threat of a compromised NoC," in *DAC*, 2014, pp. 1–6.

[20] V. Y. Raparti and S. Pasricha, "Lightweight mitigation of hardware trojan attacks in noc-based manycore computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 48.

[21] R. L. Rivest, "All-or-nothing encryption and the package transform," in *International Workshop on Fast Software Encryption*. Springer, 1997, pp. 210–218.

[22] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, and T. Shimomura, "Minimal key lengths for symmetric ciphers to provide adequate commercial security. a report by an ad hoc group of cryptographers and computer scientists," INFORMATION ASSURANCE TECHNOLOGY ANALYSIS CENTER FALLS CHURCH VA, Tech. Rep., 1996.

[23] D. Coppersmith, "The data encryption standard (des) and its strength against attacks," *IBM journal of research and development*, vol. 38, no. 3, pp. 243–250, 1994.

[24] V. Boyko, "On the security properties of oaep as an all-or-nothing transform," in *Annual International Cryptology Conference*. Springer, 1999, pp. 503–518.

[25] A. Desai, "The security of all-or-nothing encryption: Protecting against exhaustive key search," in *Annual International Cryptology Conference*. Springer, 2000, pp. 359–375.

[26] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai, "Exposure-resilient functions and all-or-nothing transforms," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 453–469.

[27] S. I. Marnas, L. Angelis, and G. L. Bleris, "All-or-nothing transforms using quasigroups," in *Proc. 1st Balkan Conference in Informatics*, 2003, pp. 183–191.

[28] S. I. Marnas, L. Angelis, and G. L. Bleris, "An application of quasigroups in all-or-nothing transform," *Cryptologia*, vol. 31, no. 2, pp. 133–142, 2007.

[29] N. J. Sloane *et al.*, "The on-line encyclopedia of integer sequences," 2003.

[30] B. D. McKay, A. Meynert, and W. Myrvold, "Small latin squares, quasigroups, and loops," *Journal of Combinatorial Designs*, vol. 15, no. 2, pp. 98–119, 2007.

[31] R. L. Rivest *et al.*, "Chaffing and winnowing: Confidentiality without encryption," *CryptoBytes (RSA laboratories)*, vol. 4, no. 1, pp. 12–17, 1998.

[32] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," 1997.

[33] M. Bellare and A. Boldyreva, "The security of chaffing and winnowing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 517–530.

[34] C. H. Gebotys and R. J. Gebotys, "A framework for security on noc technologies," in *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.* IEEE, 2003, pp. 113–117.

[35] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input prf," in *International Conference on Cryptology in India*. Springer, 2012, pp. 489–508.

[36] T. Boraten and A. K. Kodi, "Packet security with path sensitization for nocs," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1136–1139.

[37] R. P. McEvoy, M. Tunstall, C. Whelan, C. C. Murphy, and W. P. Marnane, "All-or-nothing transforms as a countermeasure to differential side-channel analysis," *International journal of information security*, vol. 13, no. 3, pp. 291–304, 2014.

[38] Y. Dodis, "Exposure-resilient cryptography," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[39] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.

[40] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," *ISPASS*, 2009.

[41] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.