Securing Network-on-Chip Using Incremental Cryptography

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

***Abstract*—Network-on-chip (NoC) has become the standard communication fabric for on-chip components in modern System-on-chip (SoC) designs. Since NoC has visibility to all communications in the SoC, it has been one of the primary targets for security attacks. While packet encryption can provide secure communication, it can introduce unacceptable energy and performance overhead due to the resource-constrained nature of SoC designs. In this paper, we propose a lightweight encryption scheme that is implemented on the network interface. Our approach improves the performance of encryption without compromising security using incremental cryptography, which exploits the unique NoC traffic characteristics. Experimental results demonstrate that our proposed approach significantly (up to 57%, 30% on average) reduces the encryption time compared to traditional approaches with negligible (less than 2%) impact on area overhead.**

***Keywords*-system-on-chip; network-on-chip; security**

I. Introduction

With the growing demand for high-performance and low-power designs, multi-core architectures are widely used in general purpose chip multiprocessors as well as special purpose system-on-chip (SoC) designs [1; 2]. The desired performance improvement of multi-core architectures cannot be fully achieved by parallelizing the applications unless an efficient interconnect is used to connect all the heterogeneous components on the chip. Network-on-chip (NoC) has become the standard interconnect solution [3]. Due to increasing SoC complexity, it is crucial to develop efficient NoC fabrics [4]. The importance of the information passing through the NoC has made it one of the focal points of security attacks. Diguet et al. have classified the major NoC security vulnerabilities as denial of service attack, extraction of secret information, and hijacking [5]. Typically, SoCs contain several assets (e.g., encryption and authentication keys, random numbers, configuration keys, and sensitive data) that reside in different Intellectual Property (IP) cores [6; 7]. Protecting communications between IPs, which involve asset propagation, is a major challenge and requires additional hardware implementing security such as on-chip encryption and authentication units. However, implementation of security features introduce area, power and performance overhead. Security engineers have to take into account these non-functional and real-time constraints while designing secure architectures to address various threats [8]. The threat model we use in this paper is as follows:

Threat Model: Figure 1 shows a typical NoC-based many-core architecture which encrypts packets transferred between IP cores. When packets are sent through the NoC, a router infected by a hardware Trojan can copy or re-route packets and send to a malicious IP sitting on the same NoC to leak sensitive information. Therefore, our model assumes that some of the IPs, as well as the routers, can be malicious. The IPs that we can trust to be non-malicious are referred to as *secure IPs*. The goal is to ensure secure communication between these secure IPs. We assume that network interfaces (NI) that connect IPs with routers are secure. This assumption is valid since the NIs are used to integrate components of an SoC and are typically built in house. A similar threat model and assumptions have been used in previous work on NoC security, proving the validity of the model [9; 10].

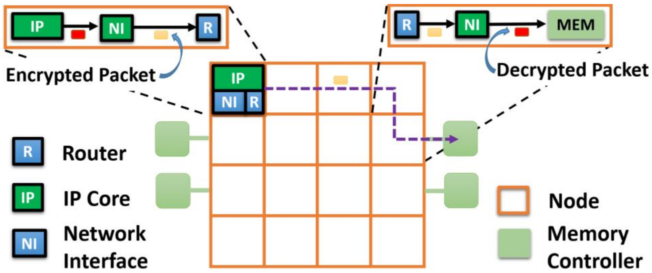


Figure 1: NoC based many-core architecture connecting IPs on a single SoC using a 4×4 Mesh topology. Each node contains an IP that connects to a router via a network interface. Communication between two IPs (in this case, a processor IP and a memory controller) is encrypted so that an eavesdropper cannot extract the packet content.

Prior research on security architectures have explored trust-zones [11; 12], lightweight encryption [13], DoS attack detection [14; 15], side channel analysis [16; 17], etc. However, none of the existing approaches have leveraged the unique traffic characteristics of an NoC to design a lightweight security architecture. In this paper, we utilize *incremental encryption* to encrypt packets in NoC. Our proposed solution takes advantage of the unique characteristics of NoC traffic, and as a result, it has the ability to construct a "lighter-weight" encryption scheme without compromising the security. Incremental cryptography has been explored in areas such as software virus protection [18] and code obfuscation [19]. To the best of our knowledge, our approach is the first attempt to utilize incremental encryption to implement a lightweight and secure NoC architecture. The goal of using incremental encryption is to design cryptographic algorithms that can reduce the effort of encryption/decryption by reusing the previously encrypted/decrypted memory fetch requests/responses rather than re-computing them from the scratch. In our framework, data is encrypted at the NI of each secure IP core. The NI is chosen to accommodate the encryption framework so that each packet can be secured before injecting into the NoC. Prior research on NoC security have proposed similar architectures where the security framework was implemented at the NI [10; 20]. Our major contributions are as follows:

- We show that consecutive NoC packets that contain memory fetch requests/responses differ only by a few bits while communicating between IP cores and memory controllers in an SoC.
- We propose a lightweight encryption scheme based on incremental cryptography that exploits the unique NoC traffic characteristics observed above.
- We show that our solution is resilient against existing NoC attacks, and it significantly improves the performance compared to state-of-the-art NoC encryption methods.

The rest of the paper is organized as follows. Section II presents prior research efforts. Section III motivates the need for our work. Section IV describe our approach for lightweight encryption. Section V presents the experimental results. Finally, Section VI concludes the paper.

II. Background and Related Work

In this section, we first provide a brief overview of concepts used in this paper and present the related research efforts to highlight the novelty of our proposed work.

A. Symmetric Encryption Schemes

A symmetric encryption scheme $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms defined as follows:

- The key generation algorithm is written as $K \leftarrow \mathcal{K}$. This denotes the execution of the randomized key generation algorithm $\mathcal{K}$ and storing the return string as K where β is the length of the key.
- The *encryption* algorithm $\mathcal{E}$ produces the *ciphertext* $C \in \{0,1\}^l$ by taking the key K and a *plaintext* $M \in \{0,1\}^l$ as inputs, where l is the length of the plaintext. This is denoted by $C \leftarrow \mathcal{E}_K(M)$.
- Similarly, the *decryption* algorithm $\mathcal{D}$ denoted by $M \leftarrow \mathcal{D}_K(C)$, takes a key K and a ciphertext $C \in \{0,1\}^l$ and returns the corresponding $M \in \{0,1\}^l$.

B. Block Ciphers

A *block cipher* typically acts as the fundamental building block of the encryption algorithm ($\mathcal{E}$). Formally, it is a function (E) that takes a β-bit key (K) and an n-bit plaintext (m) and outputs an n-bit long ciphertext (c). The values of β and n depend on the design and are fixed for a given block cipher. For every $c \in \{0,1\}^n$, there is exactly one $m \in \{0,1\}^n$ such that $E_K(m) = c$. Accordingly, E_K has an inverse block cipher denoted by E_K^{-1} such that $E_K^{-1}(E_K(m)) = m$ and $E_K(E_K^{-1}(c)) = c$ for all $m, c \in \{0,1\}^n$.

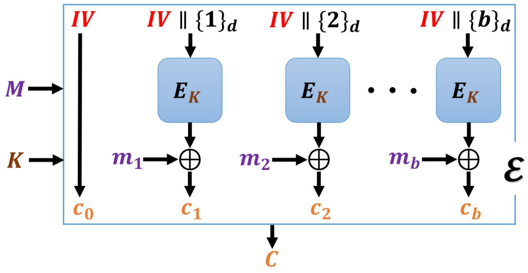


Figure 2: A block cipher-based encryption scheme using counter mode. Each block cipher (E_K) encrypts an n-bit block (m_q) and b block ciphers together encrypt the entire message M and outputs ciphertext C. This constructs $\mathcal{E}$ of the encryption scheme S.

When using block ciphers to encrypt long messages, the plaintext (M) of a given length l is divided into b substrings (m_q) where each substring is $n(=\frac{l}{b})$ bits long and n is called the *block size*. Block ciphers are used in operation modes where one or more block ciphers work together to encrypt n-bit blocks and concatenate the outputs at the end to create the ciphertext of l bits. Figure 2 shows the *counter mode (CM)* which is a popular operation mode. CM also uses an *initialization vector (IV)* which is concatenated with a d-bit value counter (e.g., if $d = 4$, $\{1\}_d = 0001$) before inputting to the block cipher. This is done to create domain separation by giving per message and per block variability. The decryption process is shown in Algorithm 1. In fact, the decryption process is the inverse of the encryption scheme shown in Figure 2.

Algorithm 1 - Decryption process of Counter Mode

Inputs: ciphertext to decrypt C
Output: plaintext corresponding to the ciphertext M
Procedure: $\mathcal{D}_K$

```
for all $q = 1, ..., b$ do
    $r_q \leftarrow E_K(IV||\{q\}_d)$
    $m_q \leftarrow r_q \oplus c_q$
$M \leftarrow m_1 \,||\, m_2 \,||\, ... \,||\, m_b$
return $M$
```

C. Incremental Cryptography Overview

Consider a scenario that involves encrypting sensitive files/documents. Once a file is encrypted initially, there may be minor changes in the original file. In such a scenario, if typical encryption is used, the previous encrypted file will be discarded and a new encryption will be performed on the modified file. However, since these changes are very small in comparison to the size of the file, encrypting the entire file again is clearly inefficient. Incremental encryption can give significant advantages in such a setup [21]. Updating an obfuscated code to accommodate patches and video transmission of images when there are minor changes between frames, are two similar scenarios [19]. Incremental encryption allows to find the cryptographic transformation of a modified input not from scratch, but as a function of the encrypted version of the input from which the modified input was derived. When the changes are small, the incremental method gives considerable improvements in efficiency.

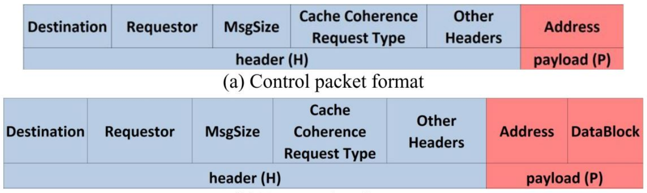
D. *Related Work*

Adhering to the tight power budgets and cost constraints, industrial and academic researchers are coming up with lightweight security primitives specific for domains such as NoC-based SoCs, RFID communication and IoT. Lightweight security architectures for NoC-based SoCs were proposed in [22; 10]. These methods try to eliminate complex encryption schemes such as AES and replace them with lightweight encryption schemes. However, these methods encrypt each block in a packet which leads to significant performance penalty. Intel introduced TinyCrypt - a cryptographic library with a small footprint which is built for resource-constrained embedded and IoT devices [13]. Several researchers have proposed other lightweight encryption solutions in the IoT domain [23]. Exploiting the unique characteristics of RFID communication, Engels et al. have proposed a low-cost encryption algorithm and a protocol [24]. None of these security architectures exploit the unique communication characteristics in NoC-based SoCs. Our approach utilizes incremental encryption to create a lightweight NoC security framework that minimizes performance overhead with minor impact on area and power.

Bellare et al. [18] used incremental encryption to encrypt files/documents undergoing minor changes. Rather than encrypting every file from scratch after each change, they proposed to encrypt only the change(s) and send it together with the previous encryption such that the encryption of the modified version can be constructed. There are fundamental challenges when using incremental encryption to encrypt packets in the NoC. In the file setup, when a file undergoes some number of modifications, every previous modification becomes redundant. In other words, intermediate steps lose their values as long as the latest version is available. However, when encrypting packets in the NoC, we cannot drop certain packets and encrypt after some modifications because each packet is important for the correct functionality of the SoC. Ideas from incremental cryptography have been adopted in other areas such as hashing and signing [21], program obfuscation [19] and cloud computing [25]. To the best of our knowledge, incremental cryptography has not been used to encrypt/decrypt NoC packets. We propose a technique that is able to use incremental encryption in the domain of NoC and can increase the efficiency of secure NoCs.

III. Motivation

The IPs use the capabilities given by the NoC to communicate with each other and to request/store data from/in memory. The packets injected into the network can be classified into two main categories - (1) control packets and (2) data packets. For example, a cache miss at an IP will cause a control packet to be injected into the network requesting for that data from the memory. The memory controller, upon receiving the request will reply back with a data packet containing the cache block corresponding to the requested address. The formats of these packets are shown in Figure 3. The NI divides the packet into flits (*fliticization*) before injecting into the network. Flits are the basic building blocks of information transfer between routers. Sensitive data of each flit is encrypted by the NI and injected into the network through the local router. Encryption process of a packet consumes time as each block has to be encrypted and concatenated to create the encrypted packet. Depending on the parameters used for the block cipher (block size, key size, number of encryption rounds, etc.), the time complexity of the process differs. If each packet is encrypted independently, it takes $z \times T$ time to encrypt all of them, where z is the number of packets and T is the average time needed to encrypt one packet.



(a) Control packet format

(b) Data packet format

Figure 3: Packet formats for control and data packets. Blue shows header (H) which is sent as plaintext. Red shows the payload (P) with sensitive data encrypted.

As discussed in Section II, the idea of incremental encryption is to develop a scheme where the time taken to encrypt an incoming packet should not be dependent on the packet size, but rather on the amount of modifications done compared to the previous packet. To explore how to use this idea in the context of NoC, we profiled the number of bit changes between consecutive packets generated by a particular IP. Figure 4 shows the number of bit differences as a percentage of memory fetch requests (control packets) when running five benchmarks (FFT, FMM, LU, RADIX, OCEAN) from the SPLASH-2 benchmark suite on the gem5 full-system simulator [26]. More details about the experimental setup is given in Section V-A. Out of the 64 bits of data to be encrypted, according to the default gem5 packet size, the maximum number of bit difference between consecutive packets was 13 bits in all benchmarks. On average, 30% of the packets differed by only one bit. This is expected since an application running on a core most likely accesses memory locations within the same memory page which differs by only a few bits.

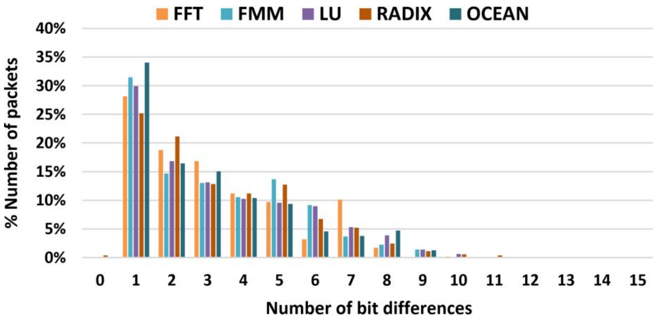


Figure 4: Number of bit differences between consecutive memory fetch requests in SPLASH-2 benchmarks.

Since encryption is done in blocks, we profiled this data assuming a block size of 16 bits [24]. In this case, up to 16 consecutive bit differences can be considered for each block, and the maximum number of blocks for 64 bits of secure data is 4. The results showed that on average, 80% of the packets differ by only one block and the other 20% differ by two blocks for the benchmarks we used. Similar to memory fetch requests, we profiled the response memory data packets as well. Since the response contains a whole cache block consisting of data modified by calculations, we don't observe the same optimization opportunity shown by memory fetch requests. However, it still shows that 15% of consecutive packets are identical. These observations show that the encryption process can be significantly optimized using incremental encryption.

IV. Incremental Encryption

This section describes our incremental encryption scheme in detail. First, we give an illustrative example to demonstrate the merit of exploiting unique traffic characteristics using incremental encryption. Then we elaborate the major components in our framework.

Illustrative example: Figure 5 shows an example on how incremental encryption can improve the performance of an NoC. It shows the encryption process of three consecutive NoC packets (each with 16 bits) using two methods (i) traditional encryption, (ii) incremental encryption. In traditional encryption, both packets are encrypted sequentially using the two 8-bit block ciphers. In incremental encryption, each packet is compared with the previous packet and only the different blocks are encrypted. Identical blocks are filled with zeros and header bits are added to indicate the changed blocks. The decryption process uses previously received packets and header information to reconstruct the new packets. Only the first packet has to be fully encrypted since there is no prior packet for comparison. This example shows a speedup of 1.43 times. However, when many packets are encrypted, the time spent to encrypt the first packet becomes negligible and as a result, we observe a significant performance improvement as shown in Section V-B. A detailed description of the methodology is given in the next three subsections.

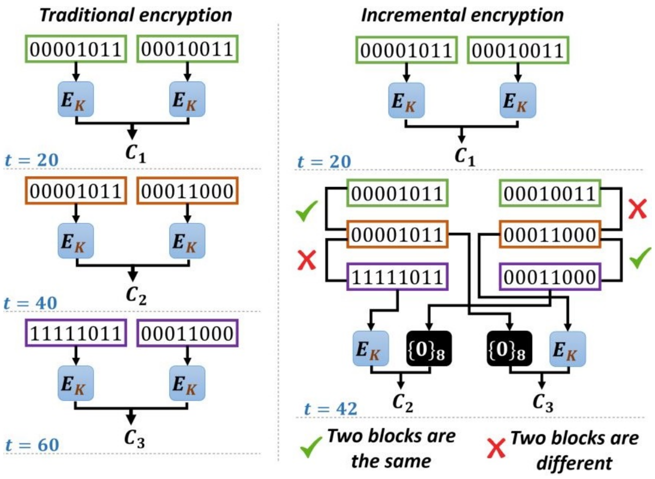


Figure 5: Illustrative example of using incremental encryption. *Assumptions: encryption takes 20 cycles for each block cipher, comparing two bit strings to identify different blocks take 1 cycle each.*

A. Overview

Figure 6 shows an overview of our proposed NoC security framework. It consists of two main components: (i) incremental crypto engine, and (ii) encryption scheme which includes the block ciphers. Each packet sent from an IP core has two main parts: (i) packet header (H) which is sent as plaintext across the network, and (ii) payload (P) which should be encrypted before sending to the network. Both header and payload are sent to the incremental crypto engine to start the incremental encryption process. We consider the payload to be divided into b blocks. For example, the 64-bit payload of a control packet will contain four 16-bit blocks ($b = 4$) numbered 1 through 4 starting from the least significant byte. Our encryption scheme uses block ciphers arranged in counter mode [27]. A detailed explanation of parameters used in our experiments is given in Section V-A.

Algorithm 2 describes our incremental encryption process. When a packet is sent from the IP core, the incremental crypto engine first identifies which blocks are different compared to the previous packet (line 3). This is done by comparing with the previous packet payload (P_{i-1}) which is stored in a register inside the NI. In our model, only two

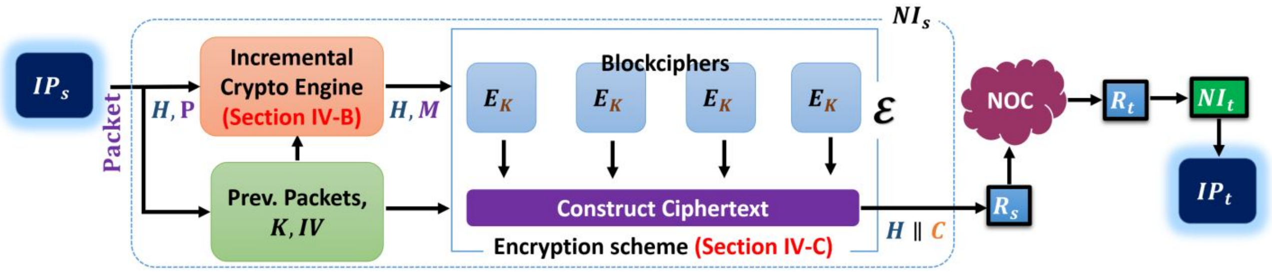


Figure 6: Overview of our proposed security framework. The packet sent from source (IP_s) goes through the encryption process implemented in the network interface (NI_s). It traverses the NoC, and NI_t of the target IP (IP_t) decrypts before forwarding the packet to IP_t.

packets are required to be stored for the two different packet types (control and data) at the sender's end. Similarly, the receiver's side also stores the most recent packet for each packet type. In addition to that, the key (K) and initialization vector (IV) for the encryption scheme are also stored by both sender and receiver IPs. Once block differences are computed, it is then sent to the encryption scheme which encrypts only the different blocks (line 4). The final ciphertext is derived from the encrypted blocks and block comparison results (line 5). Additional header bits are also computed in this step to be used by the decryption process. Finally, the header and encrypted payload are concatenated to create the final packet and injected into the network (line 6). At the destination node, the inverse process takes place. It also stores the previous packet for each packet type, and therefore, can construct the next packet using the stored packet and the incoming packet data. Since we store the previous packets in special registers, we don't have to encrypt/decrypt the full packet. We send only the changed blocks and the receiver replaces the changed blocks with its modifications to construct the new packet.

Algorithm 2 - Encryption Process

Inputs: current packet $packet_i$, previous payload P_{i-1}, key K, initialization vector IV
Output: encrypted packet consisting of header H_i and encrypted payload C_i
Procedure: $encryptPackets$

```
1: $P_i \leftarrow packet_i.payload$
2: $H_i \leftarrow packet_i.header$
3: $M_i, \delta_i \leftarrow compareBlocks(P_i, P_{i-1})$
4: $C' \leftarrow \mathcal{E}(IV, K, M_i)$
5: $C_i \leftarrow constructCipherText(C', \delta_i)$
6: return $H_i \parallel C_i$
```

The remainder of this section elaborates the major components of our NoC security framework. Section IV-B explains the $compareBlocks$ function which is implemented in the incremental crypto engine. Section IV-C presents our encryption scheme $\mathcal{E}$ and $constructCipherText$ function in Algorithm 4 and Algorithm 5, respectively.

B. Incremental Crypto Engine

The operation of the incremental crypto engine is outlined in Algorithm 3. The payload (P_i) sent from the IP core is compared with the previous payload of that type (P_{i-1}) to identify the blocks that are different (M_i). This can be implemented with a simple XOR operation in hardware (line 1). Once the bitwise differences are obtained, we split the payload into blocks (line 2) to see which blocks are different (lines 3-6). Only different blocks are sent for encryption. The incremental crypto engine also sends the different block numbers (δ_i) to build the complete ciphertext as well as to set the header bits indicating the different blocks to be used by the decryption algorithm.

Algorithm 3 - Finding Block-wise Packet Differences

Inputs: current payload P_i, previous payload P_{i-1}
Output: different blocks M_i, different block indices δ_i
Procedure: $compareBlocks$

```
1: $bitDiff \leftarrow P_i \oplus P_{i-1}$
2: $B[1], ..., B[k] \leftarrow split(bitDiff, blockSize)$
3: for all $x = 1, ..., size(B)$ do
4:     if $B[x] > 0$ then
5:         $M_i.append(B[x])$
6:         $\delta_i[x] = 1$
7: return $M_i, \delta_i$
```

C. Encryption Scheme

We use the counter mode for encryption which uses an *initialization vector* (IV), a key and the message to be encrypted as inputs and produces the ciphertext. The $IV \parallel \{q\}_d$ string, which is the standard format of the input nonce to counter mode, is used to give per message and per block variability. In our framework, it is calculated using the *sequence number* of the packet (let seq_j be the sequence number of packet P_j), a counter, and the IV as $IV \parallel seq_j \parallel q$ to identify different blocks. The block cipher ID ($q \in \{1, 2, 3, 4\}$) changes with each block cipher and the sequence number seq_j varies from packet to packet. As discussed before, the performance improvement is gained by encrypting multiple blocks in parallel. For example, if two consecutive control packets have differences in two blocks each, we can achieve twice the speedup by encrypting both at the same time compared to the traditional (non-incremental) approach where all four block ciphers will be used to encrypt each packet. Algorithm 4 shows the major steps of the encryption scheme.

Algorithm 4 - Encrypt Selected Blocks

Inputs: initialization vector IV, key K, different blocks M_i
Output: encrypted blocks C'
Procedure: $\mathcal{E}$

```
for all $q = 1, ..., 4$ do
    $seq_j \leftarrow getSequenceNumber(P_j)$
    $r_q \leftarrow E_K(IV \parallel seq_j \parallel q)$
    $C'.append(r_q \oplus M_i[q])$
return $C'$
```

C' is stored in a buffer. The final ciphertext is constructed using δ_i and C' as shown in Algorithm 5. Algorithm 5 takes the encrypted value from the buffer for the changed blocks (lines 2-3) and appends n (block size) zeros to identical blocks compared to the previous packet (lines 4-5). It ensures the construction of the same packet size, and as a result, every other functionality from fliticization to NoC traversal remains the same.

Algorithm 5 - Construct the Encrypted Payload

Inputs: encrypted blocks C', different block indices δ_i
Output: Encrypted payload C_i
Procedure: $constructCipherText$

```
1: for all $x = 1, ..., size(\delta)$ do
2:     if $\delta_i[x] > 0$ then
3:         $C_i.append(C'[x])$
4:     else
5:         $C_i.append(\{0\}_n)$
6: return $C_i$
```

To ensure the secure implementation of our approach, the generation and management of keys and nonces needs to be addressed. However, this is beyond the scope of this paper and many previous studies have addressed this problem in several ways [28; 29].

V. Experiments

In this section, we first describe the experimental setup used to evaluate our approach. Then, results are presented to show the performance gain achieved through incremental encryption by comparing it with traditional encryption. Next, we discuss the security of the proposed framework and associated overhead.

A. Experimental Setup

We validated our framework using five benchmarks chosen from the SPLASH-2 benchmark suite. Traffic traces were generated by the cycle-accurate full-system simulator - gem5 [26]. The 4×4 Mesh NoC was built on top of "GARNET2.0" model that is integrated with gem5 [30]. We modified the network interface (NI) to simulate the proposed security framework. We selected the following options to simulate architectural choices in a resource-constrained NoC.

Packet format: For control and data packet formats, we used the default GARNET2.0 implementations which allocates 128 bits for a flit. This value results in control messages fitting in 1 flit, and data packets, in 5 flits. Out of the 128 bits, 64 bits are allocated for the payload (address) in a control packet and data packets have a payload of 576 bits (64-bit address and 512-bit data). This motivated the use of 16-bit blocks to evaluate the performance of our proposed incremental encryption scheme.

Block cipher: We use an ultra-lightweight block cipher - *Hummingbird-2* as the block cipher of our encryption scheme [24]. Hummingbird-2 was chosen in our experiments mainly because it is lightweight and also, with the block size being 16, other encryption schemes can be broken using brute-force attacks in such small block sizes. However, it has been shown in [24] that Hummingbird-2 is resilient against attacks that try to recover the plaintext from ciphertext. It uses a 128-bit key and a 128-bit internal state which provides adequate security for on-chip communication. Considering the payload and block sizes, we used four block ciphers in counter mode for our encryption scheme. Each block cipher is assumed to take 20 cycles to encrypt a 16-bit block and each comparison of two-bit strings incurs a 1-cycle delay [24]. Our framework is flexible to accommodate different packet formats, packet sizes and block ciphers depending on the design requirements. For example, if a certain architecture requires 128-bit blocks, AES can be used while keeping our incremental encryption approach intact.

B. Performance Evaluation

We present the performance improvement achieved by our approach in two steps: (i) time taken for encryption (Figure 7) and (ii) execution time (Figure 8). We measured the cycles spent for encryption alone (encryption time) and total cycles executed to run the benchmark (execution time) including encryption time, using our approach as well as traditional encryption. Figure 7 shows the encryption time comparison. Our approach improves the performance of encryption by 57% (30% on average) compared to the traditional encryption schemes. The locality in data and the differences in operand values affect the number of changed blocks between consecutive packets. This is reflected in the encryption time. For example, if an application is doing an image processing operation on an image stored in memory, accessing pixel data stored in consecutive memory locations provides an opportunity for performance gain using our approach.

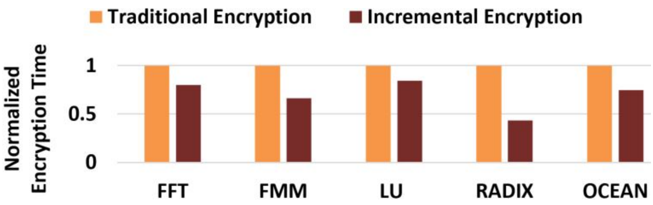


Figure 7: Encryption time comparison using traditional encryption and incremental encryption (our approach).

We also compare the total execution time using traditional encryption as well as incremental encryption. Figure 8 presents these results. When the overall system including CPU cycles, memory load/store delays and delays traversing the NoC is considered, the total execution time improves upto 10% (5% on average). Benchmarks that have significant NoC traversals such as RADIX and OCEAN show higher performance improvement (10%).

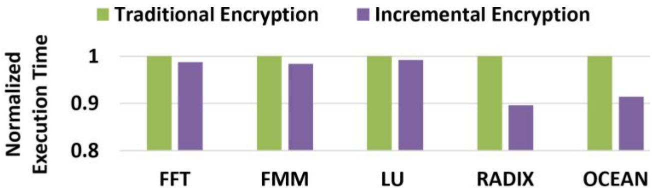


Figure 8: Execution time comparison using traditional encryption and incremental encryption (our approach).

C. Security Analysis

When discussing the security of our approach, three main components have to be considered: (i) incremental encryption, (ii) encryption scheme that uses counter mode, and (iii) block cipher.

Incremental encryption: Due to the inherent characteristics of incremental encryption, our approach reveals the amount of differences between consecutive packets. Studies on incremental encryption have shown that even though hiding the amount of differences is not possible, it is possible to hide "everything else" by using secure block ciphers and secure operation modes [18]. Attacks on incremental encryption using this vulnerability relies on the adversary having many capabilities in addition to the ones defined in the threat model. When using incremental encryption to encrypt documents undergoing frequent, small modifications as explained in Section II, it is reasonable to assume that the adversary not only has availability to the previously encrypted versions of documents but is also able to modify documents and obtain encrypted versions of the modified ones. This attack model allows the adversary to launch chosen plaintext attacks [18]. Discussing security of our approach for *known plaintext*, *chosen plaintext* and *chosen ciphertext* attacks are irrelevant in our design since the adversary does not have access to an oracle that implements the design, nor access to known plaintext/ciphertext pairs. In other words, as long as the block cipher and operation mode is secure, incremental encryption doesn't allow recovering of plaintext from the ciphertext. The same argument has been proven to hold true in previous work on incremental encryption [18; 31].

Counter mode encryption: Using our approach, each block is treated independently while encrypting, and blocks belonging to multiple packets can be encrypted in parallel. In such a setup, using the same $IV \parallel \{q\}_d$ string with the same key K can cause the "two time pad" situation. This is solved by setting the string to $IV \parallel seq_j \parallel q$ as shown in Algorithm 4. It gives per message and per block variability and ensures that the value is a nonce. Our proposed usage of counter mode adheres to the security recommendations outlined in [27].

Block cipher: As discussed above, the security of the proposed framework depends on the security of the block cipher. The security of the block cipher used in our framework, Hummingbird-2, has been discussed extensively in [24]. The first version of the Hummingbird scheme was shown to be insecure [32] and Hummingbird-2 was developed to address the security flaws. After thousands of hours of cryptanalysis, no significant flaws or sub-exhaustive attacks against Hummingbird-2 have been found [24]. Hummingbird-2 approach has been shown to be resilient against *birthday attacks* on the initialization, *differential cryptanalisys*, *linear cryptanalisys* and *algebraic attacks*. Zhang et al. presented a *related-key chosen-IV* attack against Hummingbird-2 that recovered the 128-bit secret key [33]. However, the attack requires 2^{28} pairs of plaintext to recover the first 4 bits of the key adding up to a data complexity of $\mathcal{O}(2^{32.6})$ [33]. As discussed before, launching such chosen plaintext attacks is not possible in the NoC setting. A brute force key recovery takes 2^{128} attempts which is not computationally feasible according to modern computing standards as well as for computing power in the foreseeable future.

Our proposed approach allows easy plug-and-play of security primitives. Any block size/key size/block cipher can be combined with our proposed incremental encryption approach. Note that stronger security comes at the expense of performance. Therefore, security parameters can be decided depending on the desired security and performance requirements.

D. Overhead Analysis

We implemented our proposed incremental encryption approach using Verilog to show the area overhead in comparison with the original Hummingbird-2 implementation. Our implementation is capable of assigning blocks to idle block ciphers and encrypting up to four payloads in parallel. Merger and scheduler units were implemented to ensure the correctness of final encrypted/decrypted payloads. We conducted our experiments using the Synopsys Design Compiler with 90nm Synopsis library (saed90nm). Based on our results, our proposed approach introduces less than 2% overall area overhead with respect to the entire NoC. When only the encryption unit is considered, the overhead is 15%. This overhead is caused due to components responsible for buffering and scheduling of modified blocks to idle block cipher units as well as computations related to the construction of the final result. Therefore, our proposed encryption approach has a negligible area overhead and it can be efficiently implemented as a lightweight security mechanism for NoCs. While there is a minor increase in power overhead due to the additional components, there is no penalty on overall energy consumption due to the reduction in execution time.

VI. Conclusions

In this paper, we proposed a lightweight security mechanism that improves the performance of traditional encryption schemes used in NoC while incurring negligible area and power overhead. The security framework consists of an encryption/decryption scheme that provides secure communication on the NoC. We used incremental encryption to improve performance by utilizing the unique traffic characteristics of packets observed in an NoC. We validated our framework in terms of security to prove that the performance gain is not achieved at the expense of security. Experimental results show a performance improvement of up to 57% (30% on average) in encryption time and up to 10% (5% on average) in total execution time compared to traditional encryption while introducing less than 2% overall area overhead. In the future, we plan to explore the development of an incremental authentication scheme that can be seamlessly integrated with the incremental encryption scheme to ensure data integrity.

Acknowledgment

This work was partially supported by the National Science Foundation (NSF) grant SaTC-1936040.

References

[1] S. Charles, H. Hajimiri, and P. Mishra, "Proactive thermal management using memory-based computing in multicore architectures," in *IGSC*, 2018.

[2] U. Gupta *et al.*, "Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs," *TECS*, vol. 16, no. 5s, pp. 1–20, 2017.

[3] S. Charles, C. A. Patil, U. Y. Ogras, and P. Mishra, "Exploration of memory and cluster modes in directory-based many-core cmps," in *NOCS*, 2018, pp. 1–8.

[4] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient cache reconfiguration using machine learning in noc-based many-core cmps," *TODAES*, vol. 24, no. 6, pp. 1–23, 2019.

[5] J.-P. Diguet *et al.*, "NOC-centric security of reconfigurable SoC," in *NOCS*, 2007.

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

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

[8] S. Charles, M. Logan, and P. Mishra, "Lightweight anonymous routing in noc based socs," in *DATE*, 2020.

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

[10] J. Sepúlveda *et al.*, "Towards Protected MPSoC Communication for Information Protection against a Malicious NoC," *Procedia computer science*, 2017.

[11] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *STC*, 2008.

[12] S. Charles and P. Mishra, "Lightweight and trust-aware routing in noc based socs," *ISVLSI*, 2020.

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

[14] S. Charles, Y. Lyu and P. Mishra, "Real-time detection and localization of dos attacks in noc based socs," in *DATE*, 2019, pp. 1160–1165.

[15] S. Charles, Y. Lyu, and P. Mishra, "Real-time detection and localization of distributed dos attacks in noc based socs," *IEEE TCAD*, 2020.

[16] Y. Huang *et al.*, "Scalable test generation for trojan detection using side channel analysis," *TIFS*, vol. 13, no. 11, pp. 2746–2760, 2018.

[17] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2018.

[18] M. Bellare *et al.*, "Incremental cryptography and application to virus protection," in *STOC*, 1995.

[19] S. Garg and O. Pandey, "Incremental program obfuscation," in *CRYPTO*, 2017.

[20] L. Fiorin *et al.*, "A security monitoring service for NoCs," in *CODES+ISSS*, 2008.

[21] M. Bellare *et al.*, "Incremental cryptography: The case of hashing and signing," in *CRYPTO*, 1994.

[22] K. Sajeesh and H. Kapoor, "An authenticated encryption based security framework for NoC architectures," in *ISED*, 2011.

[23] E. R. Naru *et al.*, "A recent review on lightweight cryptography in iot," in *I-SMAC*, 2017.

[24] D. Engels *et al.*, "The Hummingbird-2 lightweight authenticated encryption algorithm," in *RFIDSec*. Springer, 2011.

[25] W. Itani *et al.*, "Energy-efficient incremental integrity for securing storage in mobile cloud computing," in *ICEAC*, 2010.

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

[27] D. A. McGrew, "Counter mode security: Analysis and recommendations," *Cisco Systems, November*, 2002.

[28] B. Lebiednik *et al.*, "Architecting a secure wireless network-on-chip," *NOCS*, 2018.

[29] J. Sepulveda *et al.*, "Efficient security zones implementation through hierarchical group key management at noc-based mpsocs," *Microprocessors and Microsystems*, 2017.

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

[31] I. Mironov *et al.*, "Incremental deterministic public-key encryption," in *EUROCRYPT*. Springer-Verlag, 2012.

[32] M. J. O. Saarinen, "Cryptanalysis of hummingbird-1," in *FSE*. Springer, 2011.

[33] K. Zhang, L. Ding, and J. Guan, "Cryptanalysis of hummingbird-2," Crypto ePrint Archive, Tech. Rep., 2012.