Reconfigurable Network-on-Chip Security Architecture

SUBODHA CHARLES and PRABHAT MISHRA, University of Florida, USA

Growth of the Internet-of-things has led to complex system-on-chips (SoCs) being used in the edge devices in IoT applications. The increased complexity is demanding designers to consider several critical factors, such as dynamic requirement changes, long application life, mass production, and tight time-to-market deadlines. These requirements lead to more complex security concerns. SoC manufacturers outsource some of the intellectual property cores integrated on the SoC to untrusted third-party vendors. The untrusted intellectual properties can contain malicious implants, which can launch attacks using the resources provided by the on-chip interconnection network, commonly known as the network-on-chip (NoC). Existing efforts on securing NoC have considered lightweight encryption, authentication, and other attack detection mechanisms such as denial-of-service and buffer overflows. Unfortunately, these approaches focus on designing statically optimized security solutions. As a result, they are not suitable for many IoT systems with long application life and dynamic requirement changes. There is a critical need to design reconfigurable security architectures that can be dynamically tuned based on changing requirements. In this article, we propose a tier-based reconfigurable security architecture that can adapt to different use-case scenarios. We explore how to design an efficient reconfigurable architecture that can support three popular NoC security mechanisms (encryption, authentication, and denial-of-service attack detection and localization) and implement suitable dynamic reconfiguration techniques. We evaluate our proposed framework by running standard benchmarks enabling different tiers of security and provide a comprehensive analysis of how different levels of security can affect application performance, energy efficiency, and area overhead.

CCS Concepts: • **Computer systems organization → System on a chip**; *Multicore architectures*; *Reconfigurable computing*; • **Hardware → Network on chip**;

Additional Key Words and Phrases: Hardware security, machine learning

ACM Reference format:
Subodha Charles and Prabhat Mishra. 2020. Reconfigurable Network-on-Chip Security Architecture. *ACM Trans. Des. Autom. Electron. Syst.* 25, 6, Article 53 (August 2020), 25 pages.
https://doi.org/10.1145/3406661

1 INTRODUCTION

We are living in the era of the Internet-of-things (IoT), an era in which the number of connected devices exceeds the human population. The growth of IoT has been astounding over the last couple of decades, leading to projections of more than 50 billion connected devices by 2020 [22]. In addition to the sheer number of the devices, the complexity of these devices and different use cases in which we use them have grown remarkably. Not so long ago, our imagination was limited to

This work was partially supported by National Science Foundation (NSF) grant SaTC-1936040.
Authors' addresses: S. Charles and P. Mishra, University of Florida, 432 Newell Drive, Gainesville, FL 32611-6120; emails: {subodha96, prabhat}@ufl.edu.

1084-4309/2020/08-ART53 $15.00
https://doi.org/10.1145/3406661

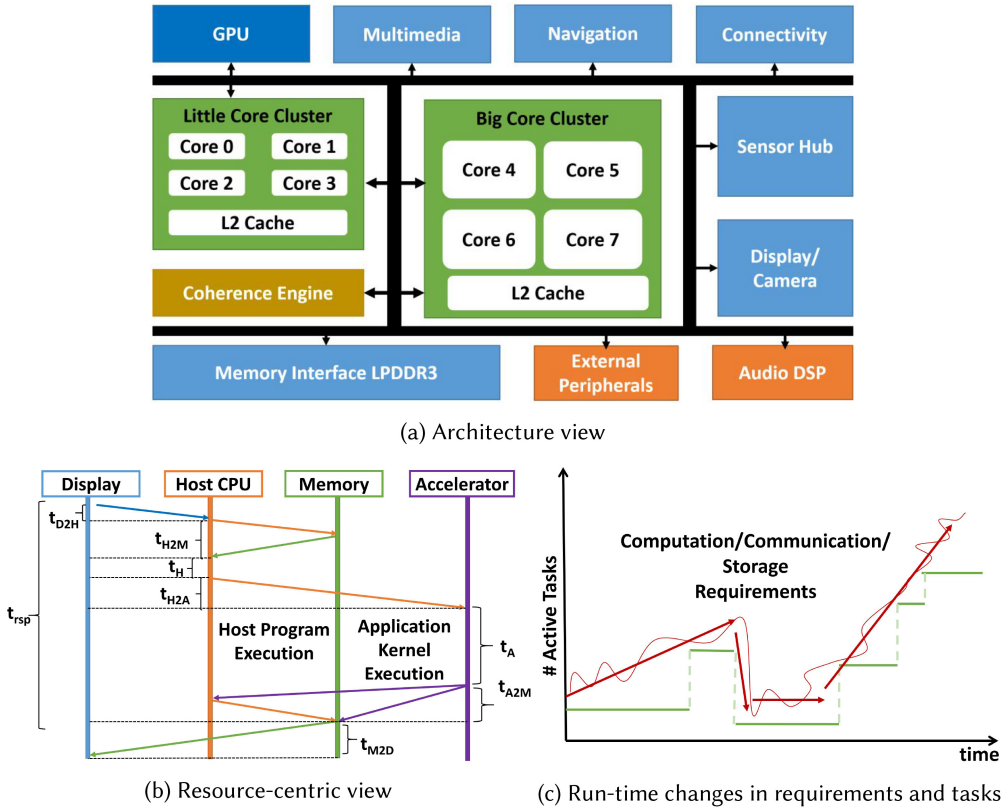


Fig. 1. Dynamic changes in IoT application characteristics: an example IoT SoC (a), application flow (b), and runtime change in requirements (c).

phones and personal computers that can run few custom applications. Today, we are developing devices that are fundamentally integrated in our day-to-day life, ranging from fitness trackers, smart homes, and smart cars, all the way to smart cities. Unlike the microcontroller-based devices in the past, even resource-constrained IoT devices consist of one or more complex system-on-chips (SoCs). The SoC components are connected by the interconnection network, known as network-on-chip (NoC) [17]. Given the rapid growth in the recent past, it is difficult to even comprehend what lies in the future. Because of this, the designers of these devices face challenges at a scale not observed before. Critical factors that affect the design choices are as follows.

Dynamic requirements and use-case scenarios. In the early days of IoT and embedded devices in general, they were intended for a single or very few use cases. The requirements and working conditions were well defined and predictable. Therefore, it was easy to make design choices to fit the requirements. For example, a device for a power-thrifty application was designed to conserve power at the cost of performance, whereas a high-performance system exhibited a different yet predictable trade-off. In comparison to that, the devices manufactured today are intended to serve general-purpose applications that are diverse and sometimes not yet defined. Therefore, it is not possible to statically optimize the devices to fit each use case. The designers should keep room for dynamic reconfigurability to address the dynamic requirements. For example, Figure 1(a) shows an IoT architecture that has been optimized during design time based on application characteristics

shown in Figure 1(b). However, this design may not be beneficial during runtime due to changes in usage scenarios, application inputs, and other parametric variations. Figure 1(c) shows that runtime reconfiguration would be useful because the computation, communication, and storage requirements can change based on specific task execution traces (phases).

Long application life. In the pre-IoT era, before devices became integrated into our everyday lives, the devices were only required to last for a few years. In the case of a phone or a personal computer, new features would come into products within a few years or even few months, and the previous models would become outdated. In contrast, if someone was building a smart house or a smart grid, it could be expected to last well over 10 years. However, the requirements of a smart system over a long life span of 10+ years can change drastically. For example, a car equipped with state-of-the-art security mechanisms will be secure in the present day but will not be secure against future attacks. The system is secure until the zero-day vulnerability is exposed. Clearly, IoT devices must be adaptable on-field to changing application requirements.

Mass production and tight time-to-market deadlines. The projection of 50 billion devices by 2020 can only happen if devices are introduced at a rapid pace. We already observe this in the market with devices being manufactured in very short periods. To achieve this, it is a common practice for SoC vendors to outsource several components of the SoC. This globally distributed supply chain of intellectual property (IP) cores make the SoCs vulnerable to trust and integrity issues. The potential space for SoC vulnerabilities is huge once we consider the seven classes of SoC-level security vulnerabilities [18]: permissions and privileges, buffer errors, resource management, information leakage, numeric errors, crypto errors, and code injection. Based on Common Vulnerabilities and Exposures (CVE-MITRE) estimates, if hardware-level vulnerabilities are removed, the overall system vulnerability will reduce by 43% [16, 18]. Therefore, securing systems against these potential threats throughout the device lifetime is a top priority.

1.1 Motivation

Summarizing the three critical factors we introduced, it is evident that securing IoT devices based on complex SoCs throughout the device lifetime among changing requirements and use-case scenarios should be considered during design time. Due to the resource-constrained nature of IoT SoCs, it is not always feasible to enforce the strongest security mechanisms. Security has to be considered among other interoperability constraints such as performance, power, and area overheads. In addition, employing the full security arsenal may not be required depending on the application characteristics and use-case scenarios. For example, consider a smart watch that is used for browsing the Internet at home, as well as in a public coffee shop. It may be okay to trust the wireless network at home and impose a lightweight security requirement in favor of a lower energy profile. However, a stronger security mechanism is necessary when communicating with the untrusted network in a coffee shop at the cost of power and performance. But if the current state of the device battery is low, it might be desirable to compromise on security and save more power to ensure application execution. The trade-off between performance and energy is also integrated in modern-day smart phones by the introduction of "power-saver" modes. Similarly, the discussion on security among other interoperability constraints is required.

According to our threat model, the security threat comes from the malicious IPs integrated on the SoC. Due to mass production and tight time-to-market deadlines, most SoC manufacturers outsource IP cores to third-party vendors. These third-party manufacturers are not always trustworthy. Their IPs might contain hardware Trojans and other malicious implants that can launch both active and passive attacks on other legitimate components on the SoC once they are activated. We call these third-party IPs *potentially malicious IPs*. Due to the distributed nature of the NoC,

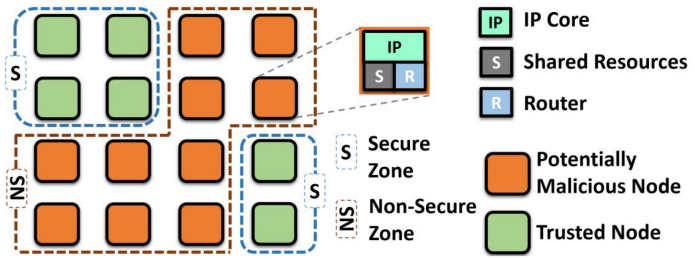


Fig. 2. Overview of a typical SoC architecture with secure and non-secure zones.

malicious IPs use resources offered by the NoC to launch attacks [19]. To capture these scenarios, we use an architecture and threat model as described in the following.

1.2 Architecture and Threat Models

In our work, we use an architecture model similar to the one shown in Figure 2. It shows an NoC-based SoC divided into secure and non-secure zones similar to the architecture proposed in the ARM TrustZone architecture [2]. The secure zone comprises IPs we can trust to not contain malicious implants (*secure IPs*), and the non-secure zone contains IPs obtained by tthird-party vendors (*potentially malicious IPs*) that cannot be trusted. An IP in one secure zone (top left) communicates secure information with a secure IP in the other zone (bottom right). Since the packets traverse through the non-secure zone, the presence of a malicious IP can pose a security threat.

Depending on increasing capabilities of malicious IPs, we divide the threats into tiers. Each tier is assumed to include the capabilities of the previous tier. For example, a malicious IP classified in tier 3 has capabilities of tiers 1 and 2 as well.

Tier 1:

- Malicious IPs can eavesdrop on the packets traversing through the network [9, 12, 13].
- Copied packets can cause information leakage.

Tier 2:

- Malicious IPs can corrupt/spoof packets. Corrupted packets can lead to the erroneous execution of programs and system failures [47].
- Spoofed packets inject new packets to the network, causing the system to malfunction.
- Packets can be rerouted to malicious IPs to leak information.

Tier 3:

- Malicious IPs can launch denial-of-service (DoS) attacks on a critical component of the SoC, causing significant performance degradation [10, 11].

We propose our reconfigurable security architecture to secure the SoC against these different capabilities of malicious IPs depending on the usage scenario. The goal is to ensure secure communication between secure IPs and to prevent any attacks. Major contributions of this work are as follows:

- We propose a reconfigurable fabric that would enable utilization of security primitives in a plug-and-play manner based on application requirements.
- We implement a tier-based security architecture that allows reconfigurable security. Solutions proposed for each tier are countermeasures for the capabilities of malicious IPs at each tier. An overview of possible attacks and corresponding countermeasures is shown in Figure 3.

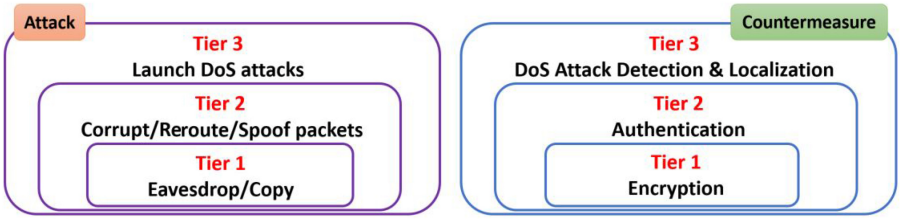


Fig. 3. Potential attacks and corresponding countermeasures in the tier-based security architecture.

- We show that the security architecture can be dynamically reconfigured based on changing application requirements.
- We implement these mechanisms on an NoC-based SoC and evaluate the efficiency of different security tiers in terms of performance, energy, and area. Then we discuss how different levels of security can be used depending on the use-case scenario.

The primary objective of the work is not to improve any of the security tiers when taken separately. Instead, the goal is to introduce a framework to integrate them together and discuss pros and cons of activating each one against the other. The remainder of the article is organized as follows. Section 2 introduces some relevant background and presents prior work in this area. Section 3 explains our reconfigurable security architecture. Section 4 presents the experimental results. Finally, Section 5 concludes the article.

2 BACKGROUND AND RELATED WORK

This section introduces some key concepts used to implement our reconfigurable security architecture. Next, it surveys related efforts in both software and hardware domains.

2.1 Block Cipher Based Symmetric Encryption

In symmetric encryption, both encryption and decryption are done using the same key ($\mathcal{K}$). Let $\mathcal{E}$ denote the encryption algorithm. If the message to be encrypted is M, the *ciphertext* C is produced by taking the key K and a *plaintext* M as inputs. This is denoted by $C \leftarrow \mathcal{E}_{\mathcal{K}}(M)$. Decryption algorithm $\mathcal{D}$ performs the inverse operation to recover the plaintext denoted by $M \leftarrow \mathcal{D}_{\mathcal{K}}(C)$. Based on input type, encryption algorithms are divided into two categories: *block ciphers* and *stream ciphers*. In block cipher based encryption schemes, the encryption algorithm comprises one or more block ciphers. Formally, a block cipher 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. To encrypt M using block ciphers, M of a given length is divided into n-bit substrings where n is called the *block size* ($n = |m|$). Each block cipher encrypts an n-bit plaintext m and concatenates the outputs at the end to create the ciphertext C corresponding to M. The arrangement of block ciphers is defined by the mode of operation used in the encryption scheme. The electronic code book (ECB) [20], cipher block chaining (CBC) [4], and counter mode (CM) [39] are three common block cipher modes of operation.

2.2 Hashing

Unlike encryption that relies on the ability to "reverse" (decrypt) the encrypted data to produce the plaintext, *hashing* data makes it extremely difficult to reverse. In fact, the security of a hash function relies on the output being computationally hard to reverse (known as pre-image resistance) and the hash function being collision resistant [54]. A hash function is a mathematical function that takes a key H and data to be hashed α as inputs and produces a *hash digest* Δ as the output denoted

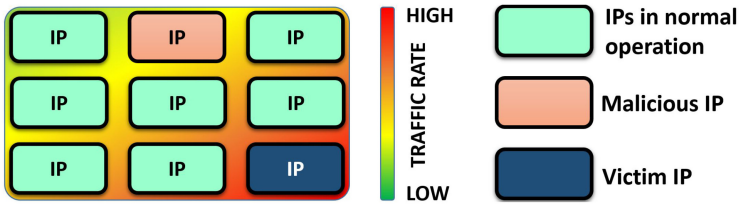


Fig. 4. Illustrative example of a DoS attack in an NoC setup with Mesh topology. The background color shows the traffic rate that relates to congestion in the NoC. Due to the large number of packets injected from the malicious IP to the victim IP, the NoC gets congested, leading to higher packet transfer delays.

by $\Delta \leftarrow \mathcal{H}(H, \alpha)$. A typical hash function produces a fixed-length digest irrespective of the size of the input data.

2.3 DoS Attacks

Figure 4 shows an illustrative example of a DoS attack scenario on an NoC. A malicious IP injects a large amount of traffic to a victim IP, causing heavy congestion in links close to the victim IPs. The attack is typically targeted at a critical NoC component such as a memory controller. Due to the congestion, legitimate memory requests from other IPs are severely delayed and the task deadlines can be violated. Previous works in this area have explored several types of DoS attacks on NoCs [24].

2.4 Related Work on Reconfigurable Software Security

The idea of reconfigurable security has been well studied in the software level. Early operating systems such as *Multics* implemented *Mandatory Access Control* schemes [3]. However, modern commercial operating systems such as Windows and Unix implement *Discretionary Access Control* systems, giving more control to the user to change access control depending on the requirements [29] at the expense of security. The kernel that is considered as the most secure among commercially available operating systems—*Security-Enhanced Linux* (SELinux)—implements *Linux Security Modules*, which give the capability to change security policies through security hooks and a policy engine. Similar to reconfigurable security in the software and firmware level, the goal of this work is to propose a reconfigurable security architecture in hardware that can be tailored during execution based on the interoperability constraints consisting of power, performance, and area.

2.5 Related Work on Reconfigurable Hardware Security

Hardware security reconfiguration has been discussed in different contexts. Hsu et al. [32] proposed a reconfigurable security architecture based on edge computing for IoT networks. The authors discussed how to interface multiple protocols (Bluetooth, ZigBee, etc.) together without changing functionality of the upper layers of a layered communication architecture. Wang et al. [52] proposed a reconfigurable encryption/decryption architecture that supported multiple cryptographic algorithms and allowed dynamic selection of algorithms depending on the requirements. Two reconfigurable encryption schemes and a reconfigurable signature scheme were proposed by Hesse et al. [31]. Their work relied on using a "common reference string" (CRS) to derive "short-term" keys from "long-term" keys. Long-term keys are kept offline, whereas short-term keys are used in cryptographic operations. In case of a leak, it is possible to update short-term keys effectively, only by "reconfiguring" the CRS, without changing long-term keys. Similar ideas have been been explored in authentication to allow execution of several authentication protocols

Table 1. Security Primitives and Corresponding Reconfigurable Parameters

Security Primitive	Reconfigurable Parameters
Encryption (tier 1)	Blockcipher, Key size, Block size, IV length
Authentication (tier 2)	Hash function, Key size, Input size
DoS attack detection and localization (tier 3)	Detection only/Detection and localization, Detection interval

dynamically [33, 50]. Other mechanisms that optimize security parameters depending on SoC behavior have also been proposed [9, 27, 28, 46]. Although the preceding approaches discuss reconfiguration of some of the cryptographic primitives independently, to the best of our knowledge, there are no prior efforts in developing a comprehensive security architecture that can enable seamless reconfiguration of a wide variety of security primitives in NoC-based SoCs.

2.6 Related Work on NoC Security

Securing the SoC using security primitives implemented in the NoC has been studied previously in several directions. Lightweight security architectures for NoC-based SoCs were proposed by Sajeesh and Kapoor [45] and Sepúlveda et al. [47]. Intel introduced TinyCrypt [34], a cryptographic library with a small footprint, for constrained embedded and IoT devices. Prior efforts on lightweight encryption have targeted both IoT [41, 51] and RF communication [21]. Pereñíguez-García and Abellán [43] presented a hash-based authentication scheme to prevent eavesdropping in wireless NoCs. Unfortunately, their mechanism incurs unacceptable performance overhead [36]. Existing countermeasures for DoS attacks tried to prevent the hardware Trojan that causes the DoS attack from triggering, using techniques such as obfuscating packets through shuffling, inverting, and scrambling [6]. If the Trojan gets triggered, techniques such as latency monitoring [35], centralized on-chip traffic analysis [23], and security verification techniques [6] tried to detect the attack. Previous work introduced a lightweight and distributed approach for DoS attack detection and localization [10], which we are using in this work to implement security in one of the tiers. None of the preceding approaches consider reconfiguration of NoC security primitives. To the best of our knowledge, the work presented in this article is the first attempt to propose a reconfigurable security architecture for NoC-based SoCs that can be dynamically reconfigured depending on use-case scenarios.

3 RECONFIGURATION OF NOC SECURITY PRIMITIVES

This section presents our proposed reconfigurable security architecture. It consists of a reconfigurable security engine (RSE), which is a dedicated IP on the SoC and security mechanisms implemented at routers and network interfaces (NIs) as outlined in Section 3.1. Although there are many security primitives, we consider three commonly utilized security primitives in NoCs (encryption, authentication, and DoS attack prevention). The security tiers are selected together with relevant parameters. We define a set of parameters that have been proposed in existing literature, as well as parameters that became reconfigurable due to our architecture. Each security tier is associated with the reconfigurable parameters as shown in Table 1.

The reconfigurable parameters in encryption and authentication are self-explanatory and have been discussed in other works [31, 50, 52]. Tier 3 allows decoupling of DoS attack detection and localization. If detection only is selected, the SoC will detect an ongoing DoS attack but not localize the malicious IP, whereas the other option enables both detection and localization. The *detection interval* defines the duration in which the detection mechanism is active. It can be always active,

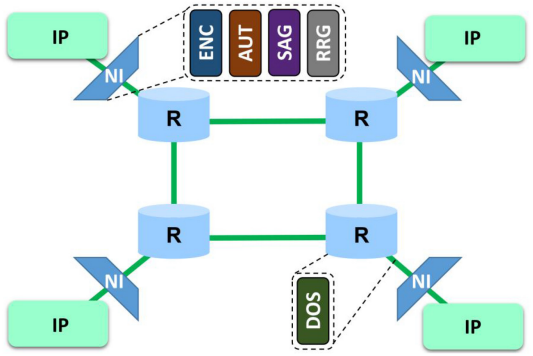


Fig. 5. Additional hardware implemented at NIs and routers to facilitate our reconfigurable security architecture. ENC, encryption; AUT, authentication; DOS, DoS attack detection.

leading to quick detection of DoS attacks, or can be periodically active to save power. The following sections describe each of these components in detail. Section 3.1 describes our reconfigurable security architecture. The next three sections present reconfigurable encryption, authentication, and DoS attack detection and localization mechanisms used in our architecture, respectively.

3.1 Reconfigurable Security Architecture

Our reconfigurable security architecture has two main parts: tier-based security countermeasures and a reconfiguration mechanism. Figure 5 shows how the security countermeasures are integrated in the NoC. The encryption and authentication tiers are integrated in the NI, whereas dedicated hardware for DoS attack detection and localization is implemented in each router and IP. Different tiers of security and their capabilities are outlined in Section 1.2. The reconfiguration mechanism decides which security tier to activate and which parameters to pass to the selected tier based on the system characteristics and security requirements. Security tiers and parameters are selected using the reconfiguration registers (RRGs) integrated into each NI that are modified by the reconfiguration mechanism.

The reconfiguration mechanism has two types of components integrated into the SoC:

(1) *Security agent*: A security agent (SAG) is integrated in each NI. SAGs monitor the network for potential security attacks and also check system characteristics such as NoC congestion and battery life through sensors.
(2) *Reconfigurable security engine*: An RSE is a dedicated IP integrated on the SoC that contains security policies and makes decisions on when to reconfigure security based on the data given by SAGs.

Figure 6 shows an overview of how the RSE is connected on the SoC, and Figure 5 shows how the SAGs are integrated into each NI. The SAGs offer three different services:

(1) Gather data about system characteristics such as battery level and NoC congestion.
(2) Pass messages received by security tiers to the RSE. For example, if an ongoing DoS attack is detected, SAGs send that information to the RSE, which can make the decision on activating the localization component of security tier 3 to localize the attack.
(3) Set the RRG in each NI to indicate which tier of security to activate according to the decisions made by the RSE.

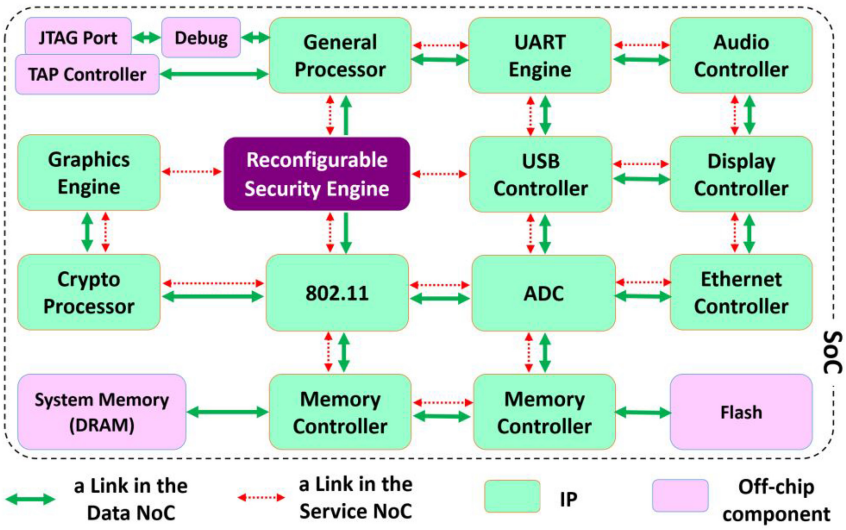


Fig. 6. Example SoC including an RSE. Figure 5 shows a zoomed-in and more detailed version of the same architecture considering only four IPs.

Algorithm 1 describes the main steps of reconfiguration. The RSE periodically pings the SAGs to gather data about system characteristics (line 2). This is called the *security heartbeat*. After gathering battery level and NoC congestion information, the RSE then decides which security tier and parameters to activate based on its security policy and passes that data to the SAGs, who set the RRGs (lines 11–14). In addition to decisions made at each security heartbeat, a SAG can also interrupt the RSE if a potential security threat is detected (line 6). The RSE will follow the same process and set the RRGs. The RSE and SAGs communicate using a separate NoC, called the *service NoC*, that facilitates all packets transferred between the RSE and SAGs without interfering with the data transferred between IPs. The IPs read the RRGs to identify which security tier to activate together with its parameters and configures security accordingly (lines 15–26). When a packet is injected into the NoC, it first goes through the security mechanisms depending on what tier is activated (lines 27–29).

This approach allows easy decoupling of the RSE, SAGs, security policy, security tiers, and reconfigurable parameters so that each component can be modified independently at design time depending on system requirements. The next three sections describe the components of tier-based security countermeasures: encryption (Section 3.2), authentication (Section 3.3), and DoS attack detection and localization (Section 3.4). A list of notations used to illustrate our approach is presented in Table 2.

3.2 Reconfigurable Encryption

To encrypt packets in real-time embedded systems, the encryption scheme should support high-speed encryption with low costs and latency. To achieve this, the operation mode of the encryption scheme must support pipelined and parallelized implementations. Furthermore, due to the nature of packets transferred and routing protocols used in the NoC, some of the packet fields, such as addresses, sequence numbers, and ports, need to be transferred in plaintext. These fields are mainly the header fields of the packet. This leads to the requirement of an authenticated encryption with associated date (AEAD) scheme. According to our threat model and proposed tier-based security model, encryption and authentication should be decoupled. Therefore, an authenticated

Table 2. Notations Used to Illustrate Our Approach

Notation	Description
$E_K(M)$	A message M encrypted using the key K
$A \parallel B$	Concatenation of two bit strings A and B
$A \oplus B$	Bitwise XOR of two bit strings A and B
$\{q\}_d$	D-bit representation of binary value q (e.g., if $d = 4$, $\{1\}_d = 0001$)
$MSB_u(S)$	Gives the most significant (leftmost) u bits of S
$len(A)$	Number of bits in A
0^u	String of u zero bits
$X \cdot Y$	Multiplication of two elements $X, Y \in GF(0^n)$, where GF corresponds to a Galois field

ALGORITHM 1: Main Steps in Security Reconfiguration

```
   /* Send security heartbeat periodically                                    */
 1 if timer > securityHeatBeatPeriod then
 2 |   send security heartbeat to all SAGs and gather data -D
 3 |   reconfigureSecurity(D)
 4 |   restartTimer()
 5 end
 6 if upon event potentialAttack == TRUE: then
 7 |   get data sent by SAG - D;                    // get data sent by SAG with interrupt
 8 |   reconfigureSecurity(D)
 9 |   restartTimer()
10 end
   /* Major steps of reconfigure security function                           */
11 Function reconfigureSecurity(D)
12 |   T_n, P_n ← selectSecurityTier(D);     // select one from T_1, T_2, T_3 and relevant parameters
13 |   send selected security tier (T_n) and parameters (P_n) to SAGs
14 |   setReconfigurationRegisters(T_n, P_n)
15 |   T_n, P_n ← readReconfigurationRegisters()
16 |   if T_n == T_1: then
17 |   |   Q ← encryption(P_n)
18 |   end
19 |   else if T_n == T_2: then
20 |   |   Q ← encryption(P_n) + authentication(P_n)
21 |   end
22 |   else if T_n == T_3: then
23 |   |   Q ← encryption(P_n) + authentication(P_n)
24 |   |   monitorDoS(P_n)
25 |   end
26 end
   /* Send packets in to the NoC                                             */
27 Function sendPackets(M)
28 |   send Q(M)
29 end
```

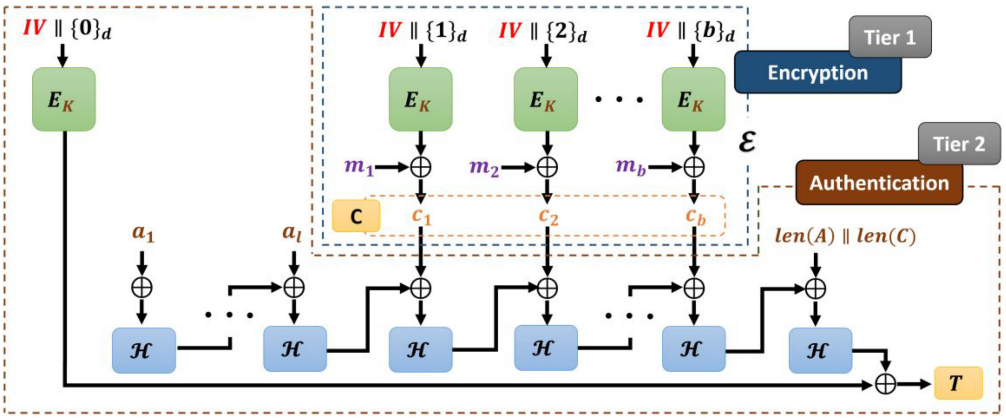


Fig. 7. Encryption and authentication in CM.

encryption scheme that allows isolation of the two stages is required. Furthermore, the architecture should allow easy plug-and-play of security primitives that allows the selection of reconfigurable parameters. To cater to these requirements, we use CM in our experiments. Figure 7 shows an overview of CM including both encryption and authentication components. It is evident from the setup that the framework supports easy decoupling of encryption and authentication, allowing activation of encryption only (tier 1) or both encryption and authentication (tier 2) through the values written in RRGs during runtime. In this section, we present how the encryption scheme in CM is implemented in the NoC, and Section 3.3 describes the NoC implementation of authentication.

Let $m_1, m_2, \ldots, m_{b-1}, m_b^*$ denote a sequence of b bit strings that construct the plaintext. Each bit string in the sequence, also known as a data block, has length n, except for m_b^* with length u where $1 \leq u \leq n$. This gives that the total length of plaintext in bits is $(b-1) \times n + u$. The ciphertext associated with this sequence follows the form $c_1, c_2, \ldots, c_{b-1}, c_b^*$ where each block is n bits long except for the final block c_b^*, which is u bits long.

The encryption algorithm is shown in Algorithm 2. Each block cipher in CM encrypts the string $IV \parallel \{q\}_d$ using a symmetric key K (line 5), where IV refers to the *initialization vector*, which is a nonce.[1] The output r_q of the block cipher is XORed with the plaintext m_q sent to that block (line 6). The final block, which can potentially have less bits (u), is XORed with the most significant u bits of the block cipher output (line 8). The outputs are concatenated to create the ciphertext of length $(b-1) \times n + u$ (line 9). The decryption process is the exact inverse of this and is omitted in this article to save space. In our experiments, we used *Galois counter mode* (GCM), which uses the same setup and AES as the block cipher together with Galoish hash as the hash function. Complete details of GCM can be found in the work of McGrew and Viega [38].

3.3 Reconfigurable Authentication

Encryption ensures that an eavesdropper cannot read the sensitive data; however, authentication is required to ensure that the adversary does not corrupt/spoof the packets. To address this, we use a hash-based message authentication code (HMAC). With HMAC, the receiver is able to verify a message by checking a tag appended to the end of the packet by the source. The receiver can recompute the original authentication tag and check whether both tags match to see that the

[1] A nonce is a random string that is distinct for each invocation of the encryption operation for a fixed key.

ALGORITHM 2: Encryption in CM

Inputs: plaintext to encrypt $M = m_1 \parallel m_2 \parallel \cdots \parallel m_b^*$
Output: ciphertext corresponding to the plaintext C
Procedure: encryption (parameters P_n = {block cipher E_K, key K, Initialization Vector IV})
for $q = 1, \ldots, b-1$ **do**
  $r_q \leftarrow E_K(IV \parallel \{q\}_d)$
  $c_q \leftarrow r_q \oplus m_q$
end
$c_b^* \leftarrow m_b^* \oplus MSB_u(E_K(IV \parallel \{b\}_d))$
$C \leftarrow c_1 \parallel c_2 \parallel \cdots \parallel c_b^*$
return C

message has not been changed during NoC traversal. The tag is computed by using a hash function that takes the message to be authenticated and a key as inputs. In our approach, since the encrypted data is used as a part of the message to be authenticated, we are following the *Encrypt-then-MAC* authentication technique, which is more secure than Encrypt-and-MAC [26].

In our AEAD scheme, the associated data was not included in the encryption process. However, associated data (A) should be used when calculating the tag. Similar to M and C, A can also be denoted as a sequence of bit strings $a_1, a_2, \ldots, a_{l-1}, a_l^*$. Each bit string in A has a length of n, except for the last block, a_l^*, with length v, where $1 \leq v \leq n$. It follows that a_l^* can be a partial block and the total length of A in bits is $(l-1) \times n + v$. If $S_q = IV \parallel \{q\}_d$, the authentication tag (T) can be calculated as

$$T = MSB_t(\mathcal{H}(H, A, C) \oplus E_K(S_0)), \tag{1}$$

where $|T| = t$ and $\mathcal{H}(H, A, C) = X_{l+b+1}$ [38]. The variable X_i for $i = 0, 1, \ldots, l+b+1$ is defined as

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus a_i) \cdot H & \text{for } i = 1, \ldots, l-1 \\ (X_{l-1} \oplus (a_l^* \parallel 0^{b-v})) \cdot H & \text{for } i = l \\ (X_{i-1} \oplus c_i) \cdot H) & \text{for } i = l+1, \ldots, l+b-1 \\ (X_{l+b-1} \oplus c_l^* \parallel 0^{b-u}) \cdot H & \text{for } i = l+b \\ (X_{l+b} \oplus (len(A) \parallel len(C)) \cdot H) & \text{for } i = l+b+1, \end{cases}$$

where $\mathcal{H}$ is the hash function that takes the hash key H as one of the inputs. The t-bit tag is then appended to the packet and injected into the network. Any tampering done to the packet will cause the tag verification at the receiver's end to fail, resulting in the packet being discarded and a retransmission of the packet from the source. This will make sure that the corrupted/spoofed packets will be discarded by NIs before they reach the IPs. The tag calculation method given in Equation (1) is according to the Galois hash function used in our experiments. Other commonly used hash functions for message authentication include SHA-256 [25] and MD5 [44].

3.4 Reconfigurable DoS Attack Detection and Localization

We implement the DoS attack detection and localization mechanism proposed by Charles et al. [10]. The previous work is a monolithic functionality, whereas in this article, we propose a reconfigurable DoS attack detection and localization algorithm. Moreover, we integrate it with reconfigurable encryption and authentication. The basic idea is to statistically analyze network traffic and to model communication patterns. Using the model, two curves are obtained that capture system characteristics. First, upper bounds of packet arrival curves (PACs) are calculated at each router and then destination packet latency curves (DLCs) are constructed at each IP. PAC

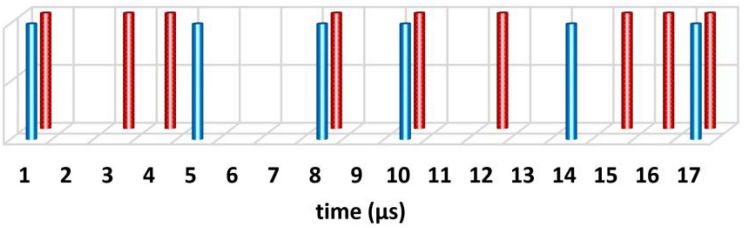


Fig. 8. Two sample event traces where the blue trace shows packet arrivals at a router under normal operation (P_r) and the red trace shows packet arrivals in the presence of a DoS attack ($\widetilde{P_r}$).

bounds are used to detect DoS attacks, and once an attack is detected, DLCs are used to localize the malicious IP.

An overview of this approach together with the parameters passed from our reconfigurable architecture is shown in Algorithm 3. The detection and localization mechanisms are implemented in such a way that localization can be disabled without affecting the detection mechanism. This is defined by the *DoSTier* parameter. In that case, an ongoing attack will be detected, but the malicious IP will not be localized. Although this approach gives better performance and energy efficiency, the malicious IP can launch the attack again unless it is diagnosed separately. Similarly, to achieve improved energy efficiency, the detection mechanism at the routers can sleep periodically. The sleep time is defined by the *detectionInterval* parameter. In such a scenario, energy efficiency will improve while compromising with delays in DoS attack detection. An SoC running tasks with soft deadlines can afford to have a DoS attack detection mechanism that is not always active. However, if there are tasks with hard deadlines, it is better to detect immediately (no sleeping) to avoid delays caused by DoS attacks. The next two sections describe the two major steps—detection and localization—and as PACs and DLCs in detail.

ALGORITHM 3: DoS Attack Detection and Localization Mechanism

```
1 Procedure: monitorDoS (parameters P_n = {DoSTier, detectionInterval})
2 if DoSTier == detectOnly then
3     detectDoS(detectionInterval); // start packet monitoring at routers and check for PAC
      bound violations
4 end
5 if DoSTier == detectAndLocalize then
6     detectDoS(detectionInterval); // start packet monitoring at routers and check for PAC
      bound violations
7     localizeDoS(); // if a potential attack is detected, initiate localization mechanism
      to pinpoint the malicious IP
8 end
```

3.4.1 DoS Attack Detection Using PAC Bounds. Upon arriving at a router (r), a packet is seen as an *event* and can be recorded with arrival curves [7]. The packet stream (P_r) comprises all of the packets that arrive at r during the execution of a particular program. Figure 8 illustrates a comparison of two different packet streams, one normal and one compromised, over the time interval $[1, 17]$. P_r (blue) shows the normal stream of packet arrivals, and $\widetilde{P_r}$ (red) shows a compromised stream with an influx in packets over the same time. For some half-closed interval, $[t_a, t_b)$, the total number of packets passing through r is called the *packet count* ($N_{p_r}[t_a, t_b)$), which is defined

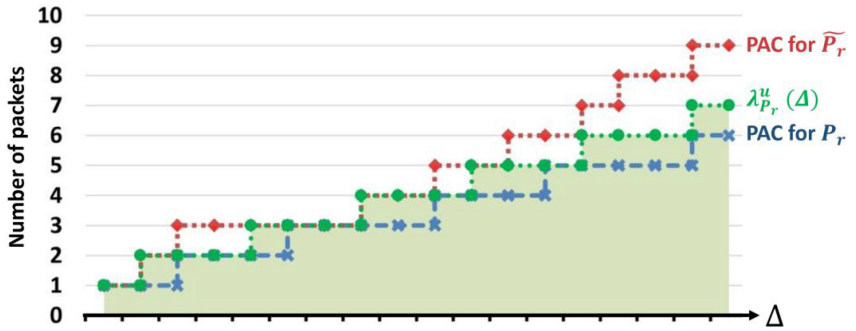


Fig. 9. Graph showing upper bound of PACs ($\lambda^u_{p_r}(\Delta)$). The green line with round markers shows the PAC bound, whereas the normal operational area is shaded in green. PACs corresponding to P_r and $\widetilde{P_r}$ are shown in blue and red, respectively.

in Equation (2). The parameters needed to calculate $N_{p_r}[t_a, t_b)$ are $N_{p_r}(t_a)$ and $N_{p_r}(t_b)$, which are the maximum number of packets before time t_a and time t_b, respectively.

$\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}$:

$$N_{p_r}[t_a, t_b) = N_{p_r}(t_b) - N_{p_r}(t_a). \tag{2}$$

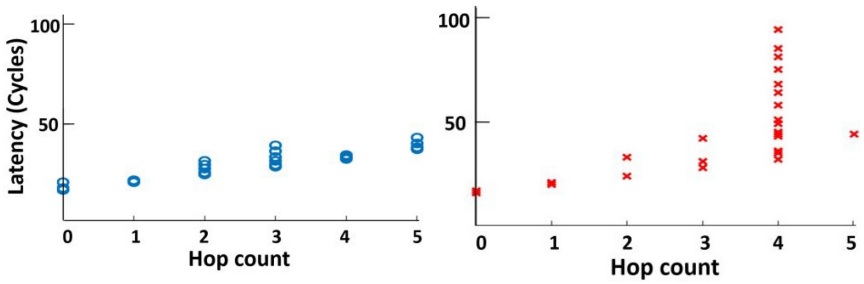
Then we construct the upper PAC bound ($\lambda^u_{p_r}(\Delta)$) for every router from the previously collected packet arrival data. To construct an upper bound, the maximum number of arrivals is necessary for any time interval $\Delta(= t_b - t_a)$. Equation (3) defines how this is done by sliding a window of length Δ over P_r to calculate the maximum number of packets arrivals within that window:

$$\lambda^u_{p_r}(\Delta) = \max_{t \geq 0}\{N_{P_r}(t + \Delta) - N_{P_r}(t)\}. \tag{3}$$

The process is repeated for several fixed Δ to construct the upper PAC bound. Once the upper PAC bound is constructed, it can be used in detecting abnormal behaviors in real time by the *leaky bucket algorithm*. Figure 9 shows a PAC bound and two PACs corresponding to P_r and $\widetilde{P_r}$ from Figure 8. It illustrates how $\widetilde{P_r}$, the compromised stream, goes beyond the shaded region indicating there is a DoS attack happening.

3.4.2 DoS Attack Localization Using PAC Bounds and DLCs. The localization method uses DLCs in addition to PAC bounds. While every router along the path constructs PACs, every destination IP constructs a DLC. Figure 10 shows two examples of DLCs with Figure 10(a) being normal operation and Figure 10(b) corresponding to an attack scenario. The DLCs capture the latency of a packet (y-axis) from source to destination D_i against the number of hops traversed by the packet from source to destination (x-axis). The distribution of latencies against each hop count follows the normal distribution, which is represented by its mean and variance. The mean and variance of the latency distribution of packets traveling k hops to reach D_i are denoted by $\mu_{i,k}$ and $\sigma_{i,k}$, respectively. The packet header holds the source and hop count information that the destination will extract for profiling. From this, the destination constructs a graph capturing the latency of packets from source to destination against the number of hops. Mean and variance for the distribution at each hop count is calculated after every packet has arrived.

Should a violation be flagged during the detection phase, the local IP attached to that router initiates the diagnosis. By referring to its DLC, it finds the packets that have suspicious (longer than usual) latencies using the parameterized $\mu_{i,k}$ and $\sigma_{i,k}$ values. The local IP then uses the source address of the delayed packets to get the congestion data from the other routers in that path. We



(a) Normal operation (b) Attack scenario

Fig. 10. Two sample DLCs constructed at an IP. Under normal operation, the variance of the distribution is small, whereas in a DoS attack scenario, it can be large. For example, the latency distribution at hop count 4 in Figure 10(b) has a large variance compared to Figure 10(a), and it helps in identifying the malicious IP.

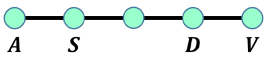


Fig. 11. Illustrative example with local IP (D), attacker IP (A), victim IP (V), and the candidate malicious IP (S) as found by D.

can only conclude that the source address of the delayed packets is a candidate malicious IP. If we conclude that the source address of the delayed packets is where the attack is originated can lead to many false positives. Therefore, the method relies on the victim pinpointing the attacker and other IPs removing the false positives. The behavior of each router during DoS attack localization is given in Algorithm 4.

We explain the behavior of the algorithm using Figure 11. The attacker IP, A, launches a DoS attack at V, and two other IPs, S and D, are located along the same congested path. Routers of D and V both will flag a potential attack (lines 1 and 2) and check the DLC for candidate malicious IPs. Even though S and D are not the attacker or the victim, packets originating from S with destination D will be delayed since they are on the congested path. Therefore, the router of S will be flagged as a candidate malicious IP by D (lines 3–7). As a result, the router of S will receive a message from the router of D indicating that its local IP is the attacker, which will cause the flag to be set to 1 (lines 8–12). However, S will receive another message from V, since S is on the path from V to A, indicating that A is the potential attacker. This will cause the flag at S to be changed to 2 (lines 13–15). The router of A will only receive the message from V, which will cause the flag to remain at 1. When the timeout occurs, the flag at S is set to 2, and therefore no action is taken. However, the router of A has a flag set to 1, and therefore a broadcast is sent indicating that A is the attacker (lines 16–20).

The overview of both DoS attack detection and localization is shown in Figure 12. The attack detection phase occurs first and is shown in the left part of the figure. The localization of the malicious IP occurs after detection as shown in the right part of the figure. The complete methodology is described in the work of Charles et al. [10]. Unlike in their work, we use the service NoC to pass messages between IPs when localizing the malicious IP to reduce the localization time.

4 EXPERIMENTAL RESULTS

In this section, we first present the experimental setup used to evaluate our framework and then show the performance and energy data for different security tiers. Finally, we discuss the area overhead and security guarantees provided by the architecture.

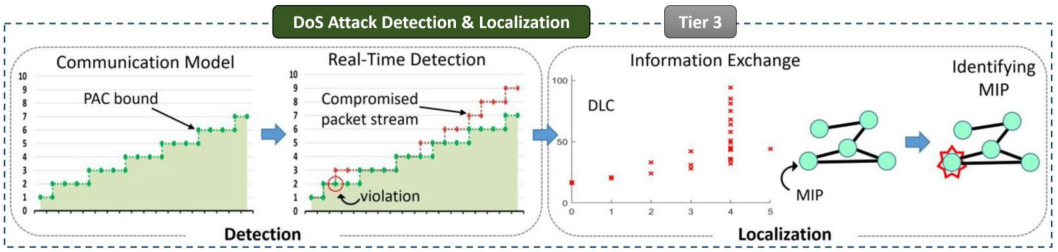


Fig. 12. Overview of the DoS attack detection and localization framework. The system behavior is captured using PAC bounds and DLCs. The curves are then used in real-time DoS attack detection and localization.

ALGORITHM 4: localizeDoS( ): Event Handlers for Routers

upon event attacked == TRUE:
send a signal to local IP

upon receiving address of the candidate malicious IP S **from local IP:**
send a query to the router of S for its congestion status
if S *is congested* **then**
  sends a *diagnostic message* $< S, D >$ to all routers in the path from S to D indicating that S is the potential attacker
end

upon receiving a diagnostic message $< S, D >$ **from port** p_i**:**
start TIMEOUT if all $flag == 0$
if S *is local IP and* $flag[p_i] == 0$ **then**
  $flag[p_i] = 1$ // local IP is the malicious IP
end
if S *is not local IP* **then**
  $flag[p_i] = 2$ // local IP is not the malicious IP
end

upon event TIMEOUT:
if $flag$ *contains 1* **then**
  broadcasting that its local IP is the attacker
  RESET
end

upon event RESET:
$flag[p_i] = 0$ for all ports p_i

4.1 Experimental Setup

Our experimental setup was built using the gem5 cycle-accurate full-system simulator [5, 14]. We modeled an 8×8 Mesh NoC-based SoC with 64 IPs. The GARNET2.0 detailed on-chip interconnection network model was used for the NoC after modifying the default garnet model to include our reconfigurable security architecture [1]. The delay for encryption/decryption and authentication was assumed to be 12 cycles [45]. GARNET2.0 uses the routing infrastructure provided by gem5's ruby memory system model. We set the number of pipeline stages in the router to be 3, and each link is assumed to consume 1 cycle to transfer a packet between neighboring routers. Each message from an IP goes through the security operations implemented at the NI and is then divided

Table 3. Reconfigurable Parameter Values Used in Our Experiments

Security Primitive	Reconfigurable Parameters
Encryption (tier 1)	AES Block cipher, 128-bit key, 128-bit block, 96-bit IV
Authentication (tier 2)	Galois hash, 128-bit key, 128-bit input
DoS attack detection and localization (tier 3)	Detection and localization both active, Detection always active without sleeping

Table 4. Execution Time Comparison in Terms of Number of Clock Cycles Across Different Security Levels Using Real Benchmarks

	FFT	RDX	FMM	LU
No-Sec	3.444E+08	2.419E+10	8.807E+09	3.684E+09
Tier 1	3.638E+08	2.669E+10	9.422E+09	3.968E+09
Tier 2	3.833E+08	2.918E+10	1.004E+10	4.251E+09
Tier 3	3.849E+08	2.939E+10	1.009E+10	4.275E+09

into flits (flow control units) before being injected into the NoC through its local router. The NoC then routes the packet depending on the routing protocol (*XY* routing in our experimental setup), and the NI at the destination performs decryption and tag validation before sending the message to the destination IP. The output statistics of the gem5 simulation were fed to the McPAT power modeling framework to obtain power consumption [37].

We tested the system using four real benchmarks (FFT, RADIX, FMM, LU) from the SPLASH-2 benchmark suite [55] and six synthetic traffic patterns (*uniform random (URD), tornado (TRD), bit complement (BCT), bit reverse (BRS), bit rotation (BRT), transpose (TPS)*). When running both real benchmarks and synthetic traffic patterns, each IP in the top (first) row of the Mesh NoC instantiated an instance of the task. Real benchmarks used eight memory controllers that provide the interface to off-chip memory, which were connected to the bottom eight IPs. As synthetic traffic patterns do not use memory controllers, the destination of injected packets were selected based on the traffic pattern. For example, uniform random selected the destination from the eight IPs at the bottom row with equal probability. Source and destination modeling was done this way to mimic the secure and non-secure zones. When simulating DoS attacks, a malicious IP that is randomly placed in the middle rows (non-secure zone) injected more packets into the NoC targeted at one of the destination IPs, which receive high traffic from legitimate requests. According to our architecture model, the IPs in the top row (secure zone) communicate with the IPs in the bottom row (secure zone) through the other six rows (non-secure zone) of IPs. Our approach will work the same for any other secure, non-secure zone selection and malicious IP placement. Table 3 shows the reconfigurable parameters selected in our experiments.

These choices were motivated by the capabilities of the simulator, as well as the lightweight nature of IoT and embedded devices [8].

4.2 Performance Results

To evaluate the execution time for each application, we simulated the setup with different security levels. Figure 13 and Table 4 show results for four levels of security when running real benchmarks:

- *No-Sec:* NoC without implementing any security.
- *Tier 1*: Tier 1 security implemented. Encryption only.
- *Tier 2*: Tier 2 security implemented. Encryption and authentication.

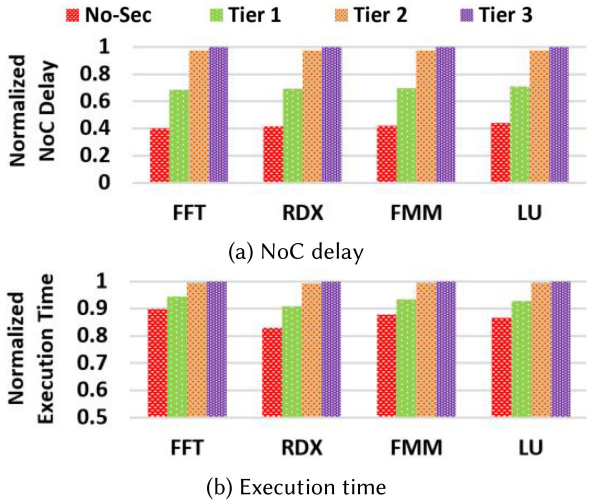


Fig. 13. NoC delay and execution time comparison across different security levels using real benchmarks.

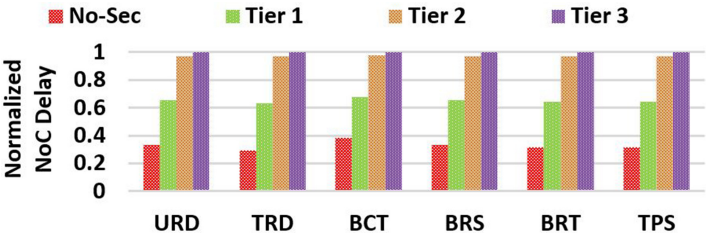


Fig. 14. NoC delay comparison across different levels of security when running synthetic traffic patterns.

- *Tier 3*: Tier 3 security implemented. Encryption, authentication, and DoS attack detection and localization.

Figure 13(a) shows NoC delay (end-to-end NoC traversal delay) of different security tiers. Compared to No-Sec, 40%, 57%, and 58% more delay is observed on average across all benchmarks in Tier 1, Tier 2, and Tier 3, respectively. Execution time is compared in Figure 13(b), and it shows a similar trend. Tier 1, Tier 2, and Tier 3 take 7%, 12.7%, and 13.2% more time to execute each simulation, respectively. The impact of security features is less in total execution time since it includes instruction execution, memory operations, and so forth, in addition to NoC traversal delay. The difference between performance in Tier 2 and Tier 3 is very small (0.5% in execution time) since the DoS attack detection mechanism can run in parallel to normal router computations once separate hardware is implemented. Charles et al. [10] reported that there is no performance overhead in their five-stage router pipeline. However, since we have implemented a three-stage router pipeline that makes the normal router computations take place faster, DoS attack detection can take a bit longer depending on crossbar contention at that time, and therefore we observe a slight delay.

The same experiments were run on synthetic traffic patterns, and results are shown in Figure 14. We can only capture NoC delay when running synthetic traffic patterns since running synthetic traffic patterns do not include instruction execution and memory operations. We observe 50%, 66%, and 67% more NoC delay on average in Tier 1, Tier 2, and Tier 3, respectively, when compared to

Table 5. Area Occupied by Security Tiers

	Tier 1	Tier 2	Tier 3	Tier 1 (Overhead)	Tier 2 (Overhead)	Tier 3 (Overhead)
Area	609696 μm^2	618473 μm^2	620228 μm^2	4.2%	5.7%	6%

Table 6. Power Consumption of Our Approach

	Tier 1	Tier 2	Tier 3	Tier 1 (Overhead)	Tier 2 (Overhead)	Tier 3 (Overhead)
Power	5,304 mW	5,387 mW	5,546 mw	3.2%	4.8%	7.9%

No-Sec. Both Figure 13 and Figure 14 show us that added security comes at the expense of performance. Therefore, the security level has to be reconfigured depending on the use-case scenario.

4.3 Overhead Analysis

This section provides details on area and power overhead of our proposed framework. It also discusses overhead associated with various components including service NoC, RSE, and security tier changes.

4.3.1 Area Overhead. Additional hardware is required to implement our reconfigurable security architecture. To evaluate this area overhead in comparison with the NoC that does not implement any security (No-Sec), we implemented the security tiers using Verilog. We modified the RTL of an open source NoC architecture [40] and conducted our experiments using Synopsys Design Compiler with a 90-nm Synopsys library (saed90nm). Results are shown in Table 5. Area overhead was calculated for each security tier. For example, area overhead for Tier 2 includes overhead introduced by encryption and authentication hardware. Since our proposed framework requires all features to be integrated so that the RSE can select which tier to activate, the total area overhead is the overhead to implement Tier 3 security, which is 6%. If a certain SoC designer decides to only integrate features in Tier 1 (encryption), overhead would be 4.2%.

If Tier 3 is implemented, in addition to the additional hardware required at routers and NIs, each IP stores and processes the DLCs. The result of $\mu_{i,k} + 1.96\sigma_{i,k}$ is calculated and stored for each hop count in the DLC as a 4-byte integer. This aggregates to a total memory space of $1 \times m \times 4$ parameters to store the DLC, where m is the maximum hop count between two IPs in the NoC. It is safe to assume that this additional memory space is negligible since the IPs typically have much more bandwidth than any other NoC component.

4.3.2 Power Overhead. The power overhead is introduced by the additional computations required to implement the reconfigurable security architecture. Compared to No-Sec, each packet injected into the network will have to go through encryption when Tier 1 is enabled. At the destination, the inverse process of decryption takes place. These processes consume extra power. Tier 2 consumes extra power for the tag computation and validation part, and Tier 3 for constructing DLCs at each IP and monitoring DoS attacks at each router. The gem5 output statistics were fed into the McPAT power modeling framework to obtain power consumption. The NoC power model in McPAT was modified according to the work done by Ogras and Marculescu [42]. Power consumption when running the four real benchmarks (FFT, RADIX, FMM, LU) were recorded, and average power consumption is compared in Table 6 together with power overhead introduced by each security tier. Tier 1 has a power overhead of 3.2%, which consists of overhead for encryption. Note that each tier includes capabilities of the tier below. In other words, 4.8% power overhead in

Tier 2 includes both encryption (3.2% in Tier 1) and authentication (1.6%) power overhead. Similarly, Tier 3 consumes 7.9% power overhead, which includes the 4.8% overhead for Tier 2 and an additional 3.1% for DoS detection and localization. The results are consistent with the previous studies on lightweight NoC encryption done by Sepúlveda et al. [47].

4.3.3 Overhead of the Service NoC. The service NoC proposed in our architecture is responsible for transferring packets intended for the following purposes:

- DoS attack localization
- Communication between SAGs and RSE
- Key distribution for encryption and authentication.

The proposal to use a separate NoC instead of using one NoC for all purposes was motivated by state-of-the-art commercial SoCs that implement multiple physical NoCs to carry different types of packets [48, 53]. The Intel Knights Landing (KNL) architecture features four parallel NoCs [48] and has been widely deployed in the Intel Xeon processor family. The Tilera TILE64 architecture comprises five parallel 2D Mesh NoCs, each used for a different purposes such as communication with main memory, communication with I/O devices, and user-level scalar operand and stream communication between tiles [53].

The trade-off here is performance versus area. When many different types of packets are used in the NoC, the packet must contain data to distinguish between those types. Existing buffer space has to be shared between packet types. Both of these concerns add performance overhead, and when scaling up to 64 IPs, the overhead becomes significant. However, contrary to intuition, additional wiring between nodes incur minimal overhead as long as the wires stay on-chip due to the advancements in fabrication processes. Furthermore, the more expensive and scarce commodity is the on-chip buffer area compared to wiring bandwidth. If virtual channels are used for different types of packets [19] and buffer space is shared, the increased buffer spaces and logic complexity will equal that of another physical network. The work of Yoon et al. [56] provides a comprehensive analysis about the trade-offs between having virtual channels and several physical NoCs [56]. Using their analysis that fits the NoC parameters we have chosen, the area and power overhead of having two physical NoCs compared to one NoC are 6% and 7%, respectively.

4.3.4 Overhead of RSE Implementation. The RSE is a dedicated IP on the SoC that decides the security tier to be used based on the security policies. The policy engine can be implemented as a finite state machine (FSM) where the security tiers are the states and state transitions happen depending on the policies. In our work, we have discussed a specific implementation where RSE decides which security tier and parameters to activate based on battery level and NoC congestion information. The implementation of such an FSM incurs negligible area and power overhead [15].

4.3.5 Overhead of Changing Security Tiers. It is worthwhile discussing what happens to the in-flight packets on the NoC during a security tier change. When changing from Tier 1 to Tier 2, the transition forces an authentication tag to be included in the NoC packets. However, the in-flight packets when the transition happens does not contain an authentication tag since they were injected when the architecture was in Tier 1. Therefore, any packet that does not contain an authentication tag will be dropped. Since we do not expect security to be reconfigured frequently, the performance overhead due to the dropped packets is negligible. In fact, security reconfiguration is expected to be less frequent compared to traditional reconfiguration techniques such as dynamic voltage scaling (DVS) or dynamic cache reconfiguration (DCR). In DCR or DVS, the reconfiguration frequency depends on the length of a phase in a task, which is in the order of milliseconds or seconds. We envision that security reconfiguration frequency will be in the order of

Table 7. Maximum Number of Packets in Flight at Any Given Time Compared to Total Number of Packets Injected When Running Each Real Benchmark

	FFT	RDX	FMM	LU
Total No. of Packets Injected	809,632	103,987,824	25,629,248	11,820,880
Maximum No. of Packets in Flight at Any Given Time	532	569	585	630
Maximum No. of Packets in Flight as a Percentage of Total No. of Packets	0.0657%	0.0005%	0.0023%	0.0053%

minutes of even hours. To quantify the performance overhead for dropping packets, we profiled the maximum number of in-flight packets at any given time when Tier 1 is active. Table 7 shows the results as a comparison of total number of packets injected when running each benchmark.

Dropping packets will not affect the accuracy of operation, since the IPs that injected the dropped packets will retransmit the requests after not receiving a response. Such retransmission mechanisms are already in place for NoC error correction protocols [30]. Note that packet dropping is not required when transitioning from Tier 2 to Tier 3 (or vice versa) due to the nature of the DoS attack detection mechanism. We have added this discussion in Section 4.3.5.

4.4 Security Analysis

In this section, we discuss the security guarantees of different security tiers.

Tier 1 implements encryption only. Therefore, the secrecy of packets is ensured, whereas the integrity of packets is not. An eavesdropper on the NoC will be unable to read the critical data in a packet unless it manages to break the cipher. The security of the cipher depends on the security of the operation mode, CM in this case, as well as the block cipher. Each block in CM is treated independently while encrypting. In such a setup, using the same $IV \parallel \{q\}_d$ string with the same key K can cause the "two time pad" situation. In our method, using a nonce as the IV for each encryption addresses this. Further security can be ensured by setting the string to $IV \parallel seq_j \parallel q$, where q corresponds to the block cipher ID and seq_j represents the sequence number of the j^{th} packet. It gives per message and per block variability and ensures that the value is a nonce. The use of $IV \parallel seq_j \parallel q$ string allows reusing the IV, and it can be reset after a certain number of encryptions. GCM uses AES as its block cipher. AES has been shown to be resistant against all known cryptographic attacks and is yet to be broken [49].

Tier 2 adds another layer of security on top of encryption by enabling authentication. This addresses the issue of data integrity. The authentication tag validation relies on the fact that unless the hash key is known, no other key and input string combination should produce the same hash digest. If this condition fails, an adversary will be able to alter the packet content, regenerate a tag for that string, and replace the existing tag with it. Then the corrupted packet will be validated as a legitimate packet. To ensure that this does not happen, the chosen hash function has to be collision resistant. Our choice of hash function—Galois hash—adheres to this criteria and is also pre-image and secondary pre-image resistant [38].

Tier 3 contributes the last layer of security of our framework—DoS attack detection and localization. To evaluate the efficiency of the approach, we ran simulations in the presence of one malicious IP placed at random in the middle rows (non-secure zone). The malicious IP injected more packets into the NoC targeted at one of the destination IPs. The packet stream periods and attack periods were selected at random. Packet steam periods were assigned a value between 2 and 6 μs at random, and attack periods were assigned a random value between 10% and 80% of the packet stream period. Experiments were conducted using the six synthetic traffic patterns and

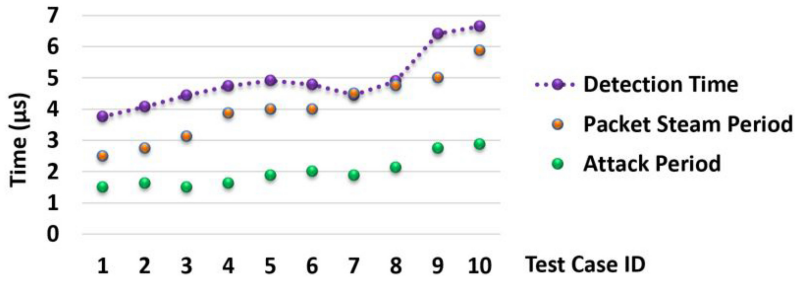


Fig. 15. DoS attack detection time for 8 × 8 Mesh topology in the presence of one malicious IP.

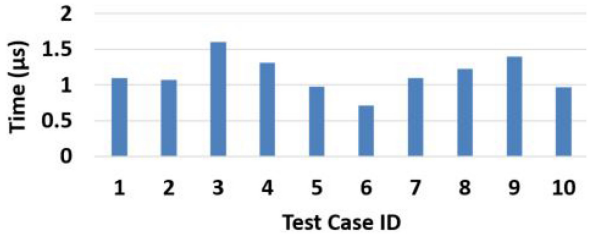


Fig. 16. DoS attack localization time for 8 × 8 Mesh topology in the presence of one malicious IP.

random placements of malicious IP launching the DoS attack. Out of the collected traces, 10 of them were selected such that the test cases include all synthetic patterns and applicable malicious IP placements. Figure 15 shows the detection time for the 10 test cases. The results show that the detection time depends on the attack period and is approximately twice the attack period. This confirms that DoS attack detection can be done in real time. The final step of Tier 3—DoS attack localization—can be done in real time as well, as shown in Figure 16. The efficiency of DoS attack localization was evaluated by measuring the time between detecting the attack and localizing the malicious IP. Figure 16 shows the results of our experiments using the same 10 test cases running synthetic traffic patterns.

The detection time and localization time both depend on characteristics of the NoC and the position of victim/malicious IPs in the NoC. The proposed method detects a DoS attack when the number of packet arrivals within a given time window exceeds the upper bound. The time taken for this to happen depends on the the constructed upper bound, packet arrival trends at routers along the path of the DoS attack, attack period, and packet stream period during normal operation. If the upper bound is tight during normal operation for a particular time window, it only takes few additional packets to violate it. Therefore, some test cases can exceed the upper bound, quickly leading to detection times being very close to the packet stream period. Some can take longer to exceed the interval, as within that time window, the upper bound was not violated.

The localization time depends heavily on the time it takes for the diagnostic packets to traverse from the IPs connected to the routers that flagged the attack to the potentially malicious IP. The localization time varies for each topology and victim/malicious IP placement. For example, if we used a Point2Point topology, localization needs diagnostic message to travel only one hop, whereas a Mesh may require multiple hops. Therefore, localization is faster in Point2Point compared to a Mesh. In general, the localization time is less compared to detection time because the localization process completes once the small number of diagnostic packets reach all of the potentially malicious IPs, whereas detection requires many packets before violating a PAC bound during runtime.

The results are consistent with the work done by Charles et al. [10], who showed that the framework is capable of detecting and localizing DoS attacks across different topologies and deterministic routing protocols. Therefore, it is a perfect fit for real-time IoT applications.

5 CONCLUSION

In this article, we presented a reconfigurable security architecture that allowed enabling/disabling of security levels (tiers) depending on the use-case scenario. Security cannot be considered alone in resource-constrained IoT devices. The interoperability constraints—performance, energy efficiency, and area—should be taken into account when deciding the level of security required. We introduce a tier-based security architecture and proposed an efficient reconfiguration mechanism that allows monitoring system characteristics and decides which security mechanism(s) to activate based on security policies. The proposed tier-based security mechanisms comprise encryption, authentication, and DoS attack detection and localization. Experimental results discussed how different tiers can affect the interoperability constraints and the security guarantees. Our reconfigurable security architecture is lightweight and provides real-time security guarantees. Therefore, it is ideal for resource-constrained IoT devices that have dynamic requirements and long application life.

REFERENCES

[1] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the 2009 International Symposium on Performance Analysis of Systems and Software*. IEEE, Los Alamitos, CA, 33–42.

[2] ARM. 2008. Security on ARM TrustZone. Retrieved July 31, 2020 from https://www.arm.com/products/silicon-ip-security.

[3] D. Elliott Bell and Leonard J. La Padula. 1976. *Secure Computer System: Unified Exposition and Multics Interpretation*. Technical Report. Mitre Corporation, Bedford, MA.

[4] Mihir Bellare, Joe Kilian, and Phillip Rogaway. 1994. The security of cipher block chaining. In *Proceedings of the 1994 Annual International Cryptology Conference*. 341–358.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[6] Travis Boraten, Dominic DiTomaso, and Avinash Karanth Kodi. 2016. Secure model checkers for network-on-chip (NoC) architectures. In *Proceedings of the 2016 International Great Lakes Symposium on VLSI (GLSVLSI'16)*. IEEE, Los Alamitos, CA, 45–50.

[7] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. 2003. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the 2003 Conference on Design, Automation, and Test in Europe (DATE'03)*, Vol. 1. 10190.

[8] Subodha Charles, Alif Ahmed, Umit Y. Ogras, and Prabhat Mishra. 2019. Efficient cache reconfiguration using machine learning in NoC-based many-core CMPs. *ACM Transactions on Design Automation of Electronic Systems* 24, 6 (2019), Article 60, 23 pages.

[9] Subodha Charles, Megan Logan, and Prabhat Mishra. 2020. Lightweight anonymous routing in NoC based SoCs. In *Proceedings of the 2020 Design, Automation, and Test in Europe Conference and Exhibition (DATE'20)*. IEEE, Los Alamitos, CA.

[10] Subodha Charles, Yangdi Lyu, and Prabhat Mishra. 2019. Real-time detection and localization of DoS attacks in NoC based SoCs. In *Proceedings of the 2019 Design, Automation, and Test in Europe Conference and Exhibition (DATE -19)*. IEEE, Los Alamitos, CA, 1160–1165.

[11] Subodha Charles, Yangdi Lyu, and Prabhat Mishra. 2020. Real-time detection and localization of distributed DoS attacks in NoC based SoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Early Access. February 10, 2020.

[12] Subodha Charles and Prabhat Mishra. 2020. Lightweight and trust-aware routing in NoC based SoCs. In *Proceedings of the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20)*.

[13] Subodha Charles and Prabhat Mishra. 2020. Securing network-on-chip using incremental cryptography. In *Proceedings of the 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20)*.

[14] Subodha Charles, Chetan Arvind Patil, Umit Y. Ogras, and Prabhat Mishra. 2018. Exploration of memory and cluster modes in directory-based many-core CMPs. In *Proceedings of the 2018 12th IEEE/ACM International Symposium on Networks-on-Chip (NOCS'18)*. IEEE, Los Alamitos, CA, 1–8.

[15] Saurabh Chaudhury, Krishna Teja Sistla, and Santanu Chattopadhyay. 2009. Genetic algorithm-based FSM synthesis with area-power trade-offs. *Integration* 42, 3 (2009), 376–384.

[16] CWE. 2017. Common Weakness Enumeration Home Page. Retrieved July 31, 2020 from https://cwe.mitre.org/.

[17] William J. Dally and Brian Towles. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Annual Design Automation Conference*. ACM, New York, NY, 684–689.

[18] DARPA. 2017. DARPA System Security Integrated Through Hardware and Firmware (SSITH). Retrieved July 31, 2020 from https://www.darpa.mil/news-events/ssith-proposers-day.

[19] Jean-Philippe Diguet, Samuel Evain, Romain Vaslin, Guy Gogniat, and Emmanuel Juin. 2007. NOC-centric security of reconfigurable SoC. In *Proceedings of the 1st International Symposium on Networks-on-Chip (NOCS'07)*. IEEE, Los Alamitos, CA, 223–232.

[20] Ibrahim F. Elashry, Osama S. Faragallah, Alaa M. Abbas, S. El-Rabaie, and Fathi E. Abd El-Samie. 2012. A new method for encrypting images with few details using Rijndael and RC6 block ciphers in the electronic code book mode. *Information Security Journal: A Global Perspective* 21, 4 (2012), 193–205.

[21] Daniel Engels, Xinxin Fan, Guang Gong, Honggang Hu, and Eric M. Smith. 2009. Ultra-lightweight cryptography for low-cost RFID tags: Hummingbird algorithm and protocol. *Centre for Applied Cryptographic Research Technical Reports* 29 (2009), 1–16.

[22] Dave Evans. 2011. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. White Paper. Cisco.

[23] Leandro Fiorin, Gianluca Palermo, and Cristina Silvano. 2008. A security monitoring service for NoCs. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, New York, NY, 197–202.

[24] Leandro Fiorin, Cristina Silvano, and Mariagiovanna Sami. 2007. Security aspects in networks-on-chips: Overview and proposals for secure implementations. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods, and Tools (DSD'07)*. IEEE, Los Alamitos, CA, 539–542.

[25] Henri Gilbert and Helena Handschuh. 2003. Security analysis of SHA-256 and sisters. In *Proceedings of the 2003 International Workshop on Selected Areas in Cryptography*. 175–193.

[26] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. 2008. How to encrypt with the LPN problem. In *Proceedings of the 2008 International Colloquium on Automata, Languages, and Programming*. 679–690.

[27] Guy Gogniat, Tilman Wolf, and Wayne Burleson. 2005. Reconfigurable security primitive for embedded systems. In *Proceedings of the 2005 International Symposium on System-on-Chip*. IEEE, Los Alamitos, CA, 23–28.

[28] Guy Gogniat, Tilman Wolf, and Wayne Burleson. 2006. Reconfigurable security support for embedded systems. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Vol. 10. IEEE, Los Alamitos, CA, 250a.

[29] Fred T. Grampp and Robert H. Morris. 1984. The UNIX system: UNIX operating system security. *AT&T Bell Laboratories Technical Journal* 63, 8 (1984), 1649–1672.

[30] Cristian Grecu, Andre Ivanov, Res Saleh, Egor S. Sogomonyan, and Partha Pratim Pande. 2006. On-line fault detection and location for NoC interconnects. In *Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS'06)*. IEEE, Los Alamitos, CA, 1–6.

[31] Julia Hesse, Dennis Hofheinz, and Andy Rupp. 2016. Reconfigurable cryptography: A flexible approach to long-term security. In *Proceedings of the 2016 Theory of Cryptography Conference*. 416–445.

[32] Ruei-Hau Hsu, Jemin Lee, Tony Q. S. Quek, and Jyh-Cheng Chen. 2018. Reconfigurable security: Edge-computing-based framework for IoT. *IEEE Network* 32, 5 (2018), 92–99.

[33] Shao-Hsiu Hung, Jui-Hung Yeh, and Jyh-Cheng Chen. 2011. sRAMP: Secure reconfigurable architecture and mobility platform. *Security and Communication Networks* 4, 4 (2011), 395–409.

[34] Intel. 2016. Using TinyCrypt Library, Intel Developer Zone. Retrieved July 31, 2020 from https://software.intel.com/content/www/us/en/develop/tools/system-studio/documentation.html.

[35] J. S. Rajesh, Dean Michael Ancajas, Koushik Chakraborty, and Sanghamitra Roy. 2015. Runtime detection of a bandwidth denial attack from a rogue network-on-chip. In *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM, New York, NY, 8.

[36] Brian Lebiednik, Sergi Abadal, Hyoukjun Kwon, and Tushar Krishna. 2018. Architecting a secure wireless network-on-chip. In *Proceedings of the 2018 12th IEEE/ACM International Symposium on Networks-on-Chip (NOCS'18)*. IEEE, Los Alamitos, CA, 1–8.

[37] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, New York, NY, 469–480.

[38] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). *Submission to NIST Modes of Operation Process* 20 (2004).
[39] David A. McGrew. 2002. *Counter Mode Security: Analysis and Recommendations*. Cisco Systems.
[40] Alireza Monemi, Jia Wei Tang, Maurizio Palesi, and Muhammad N. Marsono. 2017. ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform. *Microprocessors and Microsystems* 54 (2017), 60–74.
[41] Effy Raja Naru, Hemraj Saini, and Mukesh Sharma. 2017. A recent review on lightweight cryptography in IoT. In *Proceedings of the 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC'17)*. IEEE, Los Alamitos, CA, 887–890.
[42] Umit Y. Ogras and Radu Marculescu. 2013. *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*. Vol. 184. Springer Science & Business Media.
[43] Fernando Pereñíguez-García and José L. Abellán. 2017. Secure communications in wireless network-on-chips. In *Proceedings of the 2nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*. ACM, New York, NY, 27–32.
[44] Ronald Rivest. 1992. The MD5 Message-Digest Algorithm. Retrieved July 31, 2020 from https://tools.ietf.org/html/rfc1321.
[45] K. Sajeesh and Hemangee K. Kapoor. 2011. An authenticated encryption based security framework for NoC architectures. In *Proceedings of the 2011 International Symposium on Electronic System Design*. IEEE, Los Alamitos, CA, 134–139.
[46] Johanna Sepúlveda, Daniel Flórez, and Guy Gogniat. 2015. Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs. In *Proceedings of the 2015 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC'15)*. IEEE, Los Alamitos, CA, 1–8.
[47] Johanna Sepúlveda, Andreas Zankl, Daniel Flórez, and Georg Sigl. 2017. Towards protected MPSoC communication for information protection against a malicious NoC. *Procedia Computer Science* 108 (2017), 1103–1112.
[48] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. 2016. Knights landing: Second-generation Intel Xeon Phi product. *IEEE Micro* 36, 2 (2016), 34–46.
[49] William Stallings, Lawrie Brown, Michael D. Bauer, and Arup Kumar Bhattacharjee. 2012. *Computer Security: Principles and Practice*. Pearson Education, Upper Saddle River, NJ.
[50] L. Thulasimani and M. Madheswaran. 2010. Implementation of an energy efficient reconfigurable authentication unit for software radio. *International Journal on Computer Science and Engineering* 2, 04 (2010), 1375–1380.
[51] Muhammad Usman, Irfan Ahmed, M. Imran Aslam, Shujaat Khan, and Usman Ali Shah. 2017. SIT: A lightweight encryption algorithm for secure Internet of Things. arXiv:1704.08688.
[52] Zhu Wang, Yan Yao, Xiaojun Tong, Qinghua Luo, and Xiangyu Chen. 2019. Dynamically reconfigurable encryption and decryption system design for the Internet of Things information security. *Sensors* 19, 1 (2019), 143.
[53] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. 2007. On-chip interconnection architecture of the tile processor. *IEEE Micro* 27, 5 (2007), 15–31.
[54] Robert S. Winternitz. 1984. A secure one-way hash function built from DES. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*. IEEE, Los Alamitos, CA, 88–88.
[55] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. *ACM SIGARCH Computer Architecture News* 23, 2 (1995), 24–36.
[56] Young Jin Yoon, Nicola Concer, Michele Petracca, and Luca P. Carloni. 2013. Virtual channels and multiple physical networks: Two alternatives to improve NoC performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 12 (2013), 1906–1919.

Received August 2019; revised March 2020; accepted June 2020