Digital Watermarking for Detecting Malicious Intellectual Property Cores in NoC Architectures

Subodha Charles, *Member, IEEE,* Vincent Bindschaedler, *Member, IEEE,* and Prabhat Mishra, *Fellow, IEEE*

***Abstract*—System-on-chip (SoC) developers utilize Intellectual Property (IP) cores from third-party vendors due to increasing design complexity, cost as well as time-to-market constraints. A typical SoC consists of a wide variety of IP cores (such as processor, memory, controller, FPGA, etc.) that interact using a Network-on-Chip (NoC). This global trend of designing SoCs using third-party IPs raises serious concerns about security vulnerabilities. Since NoC facilitates communication between all IPs in an SoC, NoC is the ideal place for any hardware Trojans to hide and launch a plethora of attacks. Due to the resource-constrained nature of SoCs, developing security solutions against such attacks is a major challenge. In particular, in an eavesdropping attack, a Trojan infected router copies packets transferred through the NoC and re-routes the duplicated packets to an accompanying malicious application running on another IP in an attempt to extract confidential information. While authenticated encryption can thwart such attacks, it incurs unacceptable overhead in resource-constrained SoCs. In this paper, we propose a lightweight alternative defense based on digital watermarking techniques. We develop theoretical models to provide security guarantees. Experiments using realistic SoC models and diverse applications demonstrate that our approach can significantly outperform state-of-the-art methods.**

I. Introduction

Design considerations for roads in a city involve accessibility, traffic distribution, and handling of specific scenarios. For example, an important objective in the design of a network of roads is to ensure ease of access to popular and important places in the city such as offices, schools, parks, etc. If prominent places are all located in the same area, the roads in that area will be congested while roads in other areas will remain (relatively) empty. An architect should ensure that the traffic is as uniformly distributed as possible or the main roads have enough lanes to mitigate congestion. A System-on-chip (SoC) designer faces similar challenges when designing the communication infrastructure connecting all the SoC components, i.e., processor cores, memories, controllers, input/output, etc. As the complexity of SoCs increase, more and more Intellectual Property (IP) cores are integrated on the same SoC. State-of-the-art SoCs have hundreds of components. For example, a typical automotive SoC may include 100-200 diverse IP cores. The demand for scalable and high-throughput interconnects has made Network-on-chip (NoC) the standard interconnection solution for complex SoCs [1].

Due to time-to-market constraints, it is a common practice for manufacturers to outsource IPs to third-party vendors. Typically, manufacturers produce only a few important IPs in-house and integrate them with third-party IPs to obtain the final SoC. As a result of this distributed supply chain, it is feasible for an attacker to insert malicious implants, such as hardware Trojans, into the IPs [2], [3], [4]. A recent occurrence of a hardware security breach due to third-party vendors aiming at industrial espionage raised concerns across top US authorities [5]. The attack was facilitated by a hardware Trojan that acted as a covert backdoor and spied on computer servers used by more than 30 companies in USA.

To address this concern, we consider the following attack scenario. A hardware Trojan integrated in the NoC IP launches an attack to eavesdrop on the NoC packets. The goal is to exfiltrate information while remaining hidden, and thus the Trojan will not perform any action that would reveal its presence, such as corrupting packets to cause SoC malfunction (data integrity attacks) or degrade performance causing denial-of-service (DoS) attacks. Previous work has explored the most effective way of launching an eavesdropping attack in NoC, considering attack effectiveness and difficulty to detect the Trojan. It identified Trojan(s) inserted in NoC component(s) colluding with another malicious IP(s) as the strongest attack model. An illustrative example of this scenario is shown in Figure 1, where a hardware Trojan-infected router and an accomplice application launch an eavesdropping attack where the infected router copies packets passing through it and sends them to the accomplice application running on another malicious IP. This hardware-software collusion attack is similar to the Illinois Malicious Processor (IMP) [6]. This setting and related threat models have been the focus of [3] as well as several prior studies [7], [2], [8], [9], [10], [11].

NoC security research has proposed authenticated encryption (AE) as a solution to eavesdropping attacks [7], [11], [10]. With AE, packets are encrypted to ensure confidentiality and an authentication tag is appended to each packet to ensure integrity (and detect re-routed packets). However, the use of AE as the defense to eavesdropping attacks is sub-optimal for two reasons. First, it incurs significant performance degradation on resource-constrained devices (as we show experimentally in Section IV). Second, authentication tags may be unnecessarily complex if used only for the purpose of detecting eavesdropping attackers who seek to remain undetected as long as possible — and thus are unlikely to interfere with data integrity.

In this paper, we ask a fundamental question: *is it possible to replace authenticated encryption with a lightweight defense while maintaining security against eavesdropping attacks?* Specifically, we propose to replace the costly computation of authentication tags with a lightweight eavesdropping attack detection mechanism based on *digital watermarking*. The attack detection capabilities achieved by digital watermarking

S. Charles is with the University of Moratuwa, Colombo, Sri Lanka. e-mail: scharles@uom.lk. V. Bindschaedler and P. Mishra are with the University of Florida, Gainesville, Florida, USA.

is coupled with encryption to ensure data confidentiality. *To the best of our knowledge, this is the first work that secures NoC-based SoCs using digital watermarking.* Specifically, this paper makes the following major contributions:

- We propose a lightweight digital watermarking based security mechanism to detect eavesdropping attacks.
- We show that our proposed approach detects attacks in a timely manner and substantiate our claims using both theoretical analysis and experimental results.
- Experimental results show that our approach incurs significantly lower performance overhead compared to authenticated encryption, which makes it an ideal fit for resource-constrained SoCs.

The remainder of the paper is organized as follows. Section II highlights how our approach differs from prior related research and describes our threat model in Section III. Section IV motivates the need for our work. Section V introduces our watermarking-based attack detection method. Section VI provides theoretical guarantees on performance and security of our approach followed by experimental results in Section VII. Section VIII discusses additional security considerations. Finally, Section IX concludes the paper.

II. Related Work

We discuss related efforts in two broad categories.

A. NoC Security

State-of-the-art NoC security revolves around protecting information traveling in the network against physical, software and side channel attacks [12]. While detecting hardware Trojans in NoC IPs during design time is still in its infancy, most solutions aim to detect/mitigate the threat of hardware Trojans during runtime. To identify most prominent threats in NoC-based SoCs, we surveyed 25 related papers published in the last 10 years and categorized them into five widely studied categories of NoC security attacks: i) eavesdropping, ii) spoofing and data integrity, iii) denial-of-service, iv) buffer overflow and memory extraction, and v) side channel attacks. Results are shown in Table I.

The survey makes it evident that eavesdropping attacks are indeed one of the most widely explored threat models related to security in NoC-based SoC. The threat model used in this work is well-established and has been considered in previous work that proposed solutions to protect the SoC from a compromised NoC IP eavesdropping on data [7], [2], [8], [9], [10], [11], [3], [27]. Ancajas et al. proposed a combination of data scrambling, packet authentication and node obfuscation to prevent eavesdropping attacks [3]. In [2], a combination of threshold voltage degradation and an encoding based packet duplication detector was proposed. Charles et al. proposed to increase the difficulty of information extraction by introducing anonymous routing in the NoC [8]. Manor et al. attempted to reduce the effectiveness of hardware Trojans trying to manipulate data packets using bit shuffling and Hamming error correction codes [9]. When eavesdropping attacks are considered, packet authentication combined with encryption (authenticated encryption) is the most popular countermeasure [27], [3], [16], [13], [7], [10], [11]. Therefore, in this paper, we compare our proposed approach with the most widely adopted approach.

TABLE I: Summary of NoC security papers found in literature categorized by attack class and defense type. **Attack Class**: Eavesdropping (EAV), Spoofing/Data Integrity (SDI), Denial-of-service (DOS), Buffer Overflow and Memory Extraction (BOM) and Side Channel Attacks (SCA). **Defense Type**: Obfuscation (OBF), Detection (DET) and Localization (LOC).

Paper	Attack Class	Defense Type
Sajeesh, 2011 [13]	EAV	OBF, DET
Porquet, 2011 [14]	BOM	OBF
Wang, 2012 [15]	SCA	OBF
Kapoor, 2013 [16]	EAV	OBF, DET
Yu, 2013 [17]	SDI	OBF
Ancajas, 2014 [3]	EAV	OBF
Saeed, 2014 [18]	BOM	DET
Sepúlveda, 2015 [19]	BOM	OBF, DET
Rajesh, 2015 [20]	DOS	DET
Biswas, 2015 [21]	DOS	DET
Reinbrecht, 2016 [22]	SCA	OBF, DET
Boraten, 2016 [10]	EAV	OBF
Prasad, 2017 [23]	DOS	DET
Sepúlveda, 2017 [11]	EAV	OBF
Frey, 2017 [24]	DOS	OBF, DET
Indrusiak, 2017 [25]	SCA	OBF
Sepúlveda, 2018 [26]	DOS	DET
Hussain, 2018 [7]	EAV	DET, LOC
Kumar, 2018 [9]	EAV	OBF
Chittamuru, 2018 [27]	EAV	OBF, DET
Lebiednik, 2018 [28]	EAV	OBF
Indrusiak, 2019 [29]	SCA	OBF
Charles, 2019 [4]	DOS	DET, LOC
Raparti, 2019 [2]	EAV	DET, LOC
Charles, 2020 [8]	EAV	OBF

B. Digital watermarking

The process of hiding information related to digital data in the data itself is called digital watermarking. It has been widely used in domains such as broadcast monitoring, copyright identification, transaction tracking, and copy control. For example, in the movie industry, a unique watermark can be embedded in every movie. If the movie later gets published on the internet illegally, the embedded watermark can be used to identify the person who leaked it. Biswas et al. [30] presented a technique called circular path-based fingerprinting using fingerprint embedding against NoC IP stealing attacks. However, the threat model used in this paper - eavesdropping attacks, cannot be addressed using their approach. Network flow watermarking is one possible solution to prevent eavesdropping attacks [31]. In network flow watermarking, watermarks are embedded into the packet flow using packet content [32], timing information [33] or packet size [34]. This can be used for tracing botmasters in a botnet [35], tracing other network-based attacks [36] and service dependency detection [37]. *To the best of our knowledge, network flow watermarking has never been studied in the context of NoC.*

III. Threat Model

The global trend of distributed design, validation and fabrication has raised concerns about security vulnerabilities. Malicious implants, such as hardware Trojans, can be inserted into the RTL or into the netlist of an IP core with the intention

of launching attacks without being detected at the post-silicon verification stage or during runtime [38]. Insertion of Trojans can happen in many places of the long, distributed supply chain such as by an untrusted CAD tool or designer or at the foundry via reverse engineering [38]. As evidence of the globally distributed supply chain of NoC IPs, iSuppli, an independent market research firm, reports that the FlexNoC on-chip interconnection architecture [39] is used by four out of the top five Chinese fabless semiconductor OEM (original equipment manufacturer) companies [40]. In fact, Arteris, the company that developed FlexNoC, achieved a sales growth of 1002% over a three-year time period through IP licensing [40]. Therefore, there is ample opportunity for attackers to integrate hardware Trojans in the NoC IP and compromise the SoC. NoC IPs are ideal candidates to insert hardware Trojans due to several reasons: i) the complexity of NoC IPs makes it extremely difficult to detect hardware Trojans during functional verification as well as runtime [2], ii) extracting data from NoC packets allows attackers to obtain confidential information without relying on memory access or hacking into individual IPs, and iii) the distributed nature of NoC components across the SoC makes it easier to launch attacks.

We focus on eavesdropping attacks, also known as snooping attacks, which pose a serious threat to applications running on many-core SoCs. IPs that are integrated on the same SoC use the NoC IP when communicating through message passing as well as through shared memory. For example, the Intel Knights Landing architecture prompts memory requests/responses from cores to traverse the NoC for shared cache look-ups and for off-chip memory accesses [1]. Therefore, eavesdropping on data transferred through the NoC allows adversaries to extract confidential information.

Adversarial model: In this paper, we consider an adversary consisting of a hardware Trojan-infected router and a colluding malicious application running on an IP. The goal of the adversary is to exfiltrate confidential information by observing NoC traffic *without being detected.* Remaining hidden is key for the adversary to exfiltrate as much information as possible. Because the adversary must remain hidden, we assume that the adversary does not interfere with the normal operation of the NoC. For example, this means that the adversary does not modify the content of packets (attack on integrity) or cause large delays in processing of packets (denial-of-service) as either would likely lead to detection.

Attack scenario: Eavesdropping attacks by malicious NoC IPs rely on the hardware Trojan creating duplicate packets with modified headers (specifically, destination address in the header) and sending them into the NoC for an accomplice application to receive them [3], [2]. Figure 1 shows an illustrative example. We consider a commonly used 2D Mesh NoC topology where IPs are connected to the NoC, more specifically to the router, via a network interface (NI). When the NI receives a message from the local IP, the message is packetized and injected into the network.[1] Packets injected

[1]Most NoCs facilitate flits, which is a further breakdown of a packet used for flow control purposes. We stick to the level of packets for the ease of explanation as our method remains the same at the flit level as well.

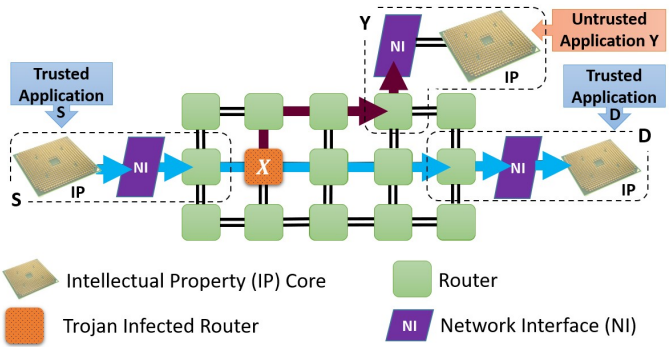


Fig. 1: Illustration of an eavesdropping attack through colluding hardware and software. A hardware Trojan integrated in a router (X) copies packets passing through it and sends them to a malicious application running on an IP (Y). An NI and an IP core are connected to each router. (For clarity, only three such pairs are shown.)

into the NoC are routed using the hop-by-hop, turn-based XY routing algorithm and received by the destination router. The NI then combines the packets to form the message which is passed to the intended destination IP. In our example (Figure 1), two trusted applications running in nodes S and D are communicating with each other, and an eavesdropping attack is launched to steal confidential information. The attack is carried out by two main components: i) a Trojan-infected router, and ii) an IP running a malicious application. The malicious router (X) copies packets passing through it and sends them to the IP running the malicious program at node Y, which reads the confidential information. To facilitate this attack, several steps should be carried out by the attacker. First, the hardware Trojan is inserted by the third-party NoC IP provider during design time. The Trojan is designed such that it can act upon commands sent by the malicious application. Once the SoC is deployed, the malicious application sends commands at a desired time to launch the attack. The Trojan then starts copying and sending packets to the malicious application. The malicious application can also send commands to pause the attack to avoid being detected.

Figure 2 shows a block diagram of a router design infected with the Trojan that launches the attack described in our threat model [3]. The Trojan copies packets arriving at the input buffer, changes the header information so that the new destination of the packet is where the malicious application is (node Y according to our illustrative example) and inject the new packet back to the input buffers so that it gets routed through the NoC to reach Y. The Trojan does not tamper with any other part of the packet, except for the header to re-route the packet, due to two reasons: i) the goal is to extract information, so corrupting data defeats the purpose, and ii) corrupting data increases chances of the Trojan getting detected. Since the original packet is not tampered with and is routed to the intended destination D, the normal operation of the SoC is preserved. The Trojan also has a very small area and power footprint. Ancajas et al. [3] used a similar threat model and reported 4.62% and 0.28% area and power overhead, respectively, when compared with the router design

without the Trojan. The performance overhead for routing packets to the malicious application is less than 1% [3].

Note that the adversarial model does not assume that the processing IP is malicious, rather it assumes that the IP is running a malicious program. While there are a wide variety of IP trust validation approaches [38], [41], they are not designed for IPs running malicious programs. Therefore, the likelihood of the Trojan being detected is very small unless additional security mechanisms (such as the one proposed in this paper) are implemented. The goal of the malicious program is to eavesdrop on the packets to attack on-chip communication security, therefore, the defense mechanism needs to be at the network layer [42], which is the focus of this paper.

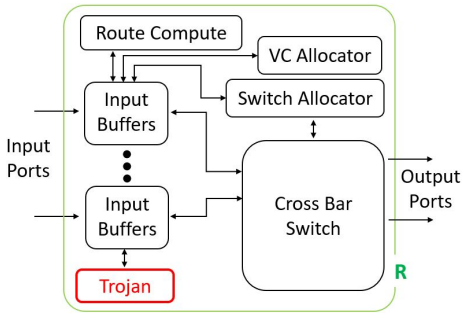


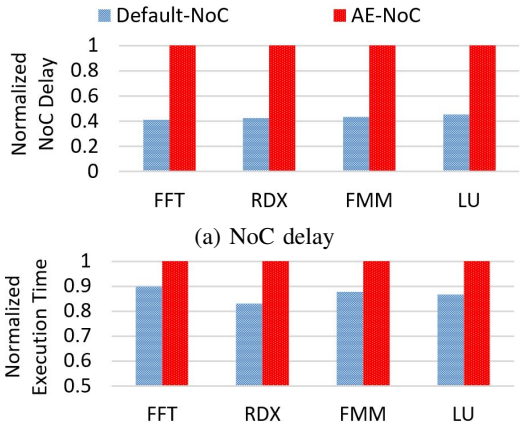
Fig. 2: Router infected with a hardware Trojan.

IV. Motivation

As explained in Section II, AE is a widely accepted countermeasure against eavesdropping attacks. Encryption provides packet confidentiality and authentication is capable of detecting re-routed packets. Since the header is modified by the hardware Trojan in order to re-route the packet to the malicious application, the authentication tag validation fails and the attack is detected. To analyze the performance overhead introduced by an AE scheme, we ran FFT, RADIX (RDX), FMM and LU benchmarks from the SPLASH-2 benchmark suite [43] on an 8×8 Mesh NoC-based SoC with 64 IPs using the gem5 simulator [44] considering two scenarios:

- **Default-NoC**: Bare NoC that does not implement encryption or authentication.
- **AE-NoC**: NoC that uses an authenticated encryption.

More details about the experimental setup is given in Section VII-A. Results are shown in Figure 3. A 12-cycle delay was assumed for encryption/decryption and authentication tag calculation when simulating AE-NoC according to the evaluations in [16]. The values are normalized to the scenario that consumes the most time. AE-NoC shows 59% (57% on average) increase in NoC delay (average NoC traversal delay for all packets) and 17% (13% on average) increase in execution time compared to the Default-NoC. The overhead for security has a relatively lower impact on execution time compared to the NoC delay since the execution time also includes the time for executing instructions and memory operations (in addition to NoC delay). NoC delay in Default-NoC case is caused by delays at routers, links and the NI. In AE-NoC, in addition to those delays, encryption/decryption delays and authentication tag calculation/validation delays are added to each packet.



(a) NoC delay

(b) Execution time

Fig. 3: NoC delay and execution time comparison across different levels of security for four SPLASH-2 benchmarks.

Additional delays are due to complex encryption/decryption operations and hash calculations for authentication.

When security is considered, Default-NoC leaves the data totally vulnerable to attacks, whereas AE-NoC ensures confidentiality and data integrity. For systems with real-time requirements, an execution time increase of 17% to accommodate a security mechanism is unacceptable. Furthermore, validating the authentication tag for each packet contributes to the SoC power consumption. Since the Trojan is rarely activated and only the packet header is modified (packet data is not corrupted) to avoid detection, authenticating each packet becomes inefficient in terms of both performance and power consumption [7]. Clearly, authenticating to detect re-routed packets introduce unnecessary overhead. It would be ideal if the security provided by AE-NoC could be achieved while maintaining the performance of Default-NoC. However, in resource-constrained environments, there is always a trade-off between security and performance.

In this paper, we propose a novel digital watermarking-based security mechanism that incurs minimal overhead while providing high security. Our approach replaces authentication by watermarking. Encryption is used to ensure data confidentiality. Our method achieves a better trade-off than: (1) no authentication that is vulnerable to credible Trojan attacks, and (2) authenticated encryption, which incurs performance degradation limiting their use in real-time applications.

V. NoC Packet Watermarking

In this section, we first present a few key definitions and concepts used in our proposed watermarking construction. We then describe our lightweight eavesdropping attack detection mechanism based on digital watermarking.

A. Definitions

In this section, we introduce two important definitions that would be used in the rest of the paper.

1) Hoeffding's Inequality: Let $\{X_1, ..., X_n\}$ be a sequence of independent and bounded random variables with $X_i \in [a, b]$ for all i, where $-\infty < a \leq b < \infty$. Then;

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}(X_i - \mathbb{E}[X_i])\right| \geq t\right] \leq e^{\left(-\frac{2nt^2}{(b-a)^2}\right)}$$

for all $t \geq 0$ [45]. By Hoeffding's Lemma, which says if $X_i \in [a, b]$ then $\mathbb{E}\left[e^{\lambda X}\right] \leq e^{\lambda^2 (b-a)^2/8}$ for any $\lambda \geq 0$, a random variable bounded in $[a, b]$ is sub-Gaussian with variance proxy $\sigma^2 = \frac{(b-a)^2}{4}$. Therefore;

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}(X_i - \mathbb{E}\left[X_i\right])\right| \geq t\right] \leq e^{\left(-\frac{nt^2}{2\sigma^2}\right)} \quad (1)$$

2) Bounds for Binary Codes: Let $\mathcal{C}$ be a binary code of length w, size M (i.e., having M codewords) and minimum Hamming distance δ between any two codewords denoted by (w, M, d). The distance distribution of $\mathcal{C}$ can be calculated as;

$$B_i = \frac{1}{M}\sum_{c \in \mathcal{C}} |c' \in \mathcal{C} : \mathcal{D}(c, c') = i|, 0 \leq i \leq n$$

It is clear that $B_0 = 1$ and $B_i = 0$ for $0 < i < d$ [46].

Let $A(w, d)$ represent the maximum number of codewords M in any binary code of length w and minimum Hamming distance d between codewords. Finding optimum $A(w, d)$ for a given w and d is an NP-Hard problem [47]. However, exact solutions are known for few combinations of values and in the general case, upper and lower bounds of the maximum number of codewords are known [48].

B. Overview

We call the flow of packets sent from one IP (source) to another IP (destination), a *packet stream*. Our detection mechanism relies on the following assumptions about the architecture and threat model.

- The Trojan does not tamper with the legitimate packet content as this may reveal its presence (Section III). The Trojan only modifies the header of duplicated packets to change the destination (data fields of the duplicated packets are not tampered with) and it allows the legitimate packets to pass as usual.
- Packets are not dropped by intermediate routers and the order of packets in a packet stream is kept constant. This is reasonable as deadlock and livelock free XY routing is used together with FIFO buffers [2].
- When the attacker injects copied packets into the NoC, all the packets can get delayed due to congestion. While this delay is random, the maximum delay is bounded. We explore this assumption in detail in Section VI-B.

Our proposed approach is to embed a unique watermark into every packet stream. Figure 4 shows an overview. We propose to include the watermark encoder and decoder at the NI of each node. It is reasonable to assume that the NI can be trusted since it acts as the interface between all the IPs in the SoC and the NoC IP, and is typically designed in-house [16], [8]. The NI at source S encodes the watermark and the NI at destination D decodes it to identify that the packet stream is valid, or in other words, the packets in the packet stream are intended to be received by D. This process is followed by each source/destination pair in the NoC. In case of an attack, the watermark decoded by the NI of the receiving node (node Y according to our illustrative example), will be invalid and a potential attack is flagged. To ensure this behavior, the watermarking mechanism must have the following characteristics:

1) The watermark is unique to each packet stream.
2) There is a shared secret between S and D, which is "hard" for any other node to guess or deduce.

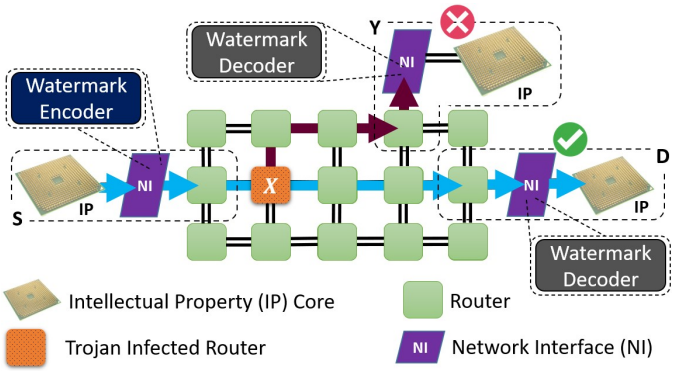


Fig. 4: Overview of the watermarking scheme where the watermark encoder and decoder are implemented at the NI.

In addition to watermarking, we rely on encryption/decryption modules implemented at the NIs. The watermark is embedded in the encrypted packets and is decoded before the decryption process. Encrypting packets is required to provide data confidentiality during packet transfers and due to the nature of our watermarking scheme that allows the malicious application to receive some packets before detecting the attack. Proposing an encryption mechanism is beyond the scope of this paper and several previous work have already proposed NoC-based SoC architectures with encryption/decryption modules implemented at the NI [16], [13], [8]. Our proposed watermarking scheme can be implemented on top of those solutions. The performance improvement is achieved by replacing the authentication scheme with our lightweight digital watermarking scheme. The following sections describe our approach in detail. First, we outline the concept behind probabilistic NoC packet watermarking (Section V-C), and then discuss the operation of the watermark encoder and decoder in detail (Section V-D). Finally, we outline an effective method for managing secrets shared between nodes (Section V-E).

C. Probabilistic Watermarking Concept

The watermark ω_{SD} is embedded by the NI of S before the packets are injected into the NoC. We use a timing-based watermark (as opposed to size or content-based) for three reasons; (i) timing alterations are harder to detect by an attacker, (ii) it allows a lightweight implementation as it is easy to manipulate, and (iii) it does not alter the packet content allowing encryption schemes to be implemented together with watermarking. The watermark is embedded by slightly delaying certain packets in the stream. If ω_{SD} is unique, it should be correctly decoded at the NI of destination D with high probability. In contrast, the probability of decoding ω_{SD} as valid at any other NI should be very low.

Given n packets of a packet stream P_{SD} such that;

$$P_{SD} = \{p_{SD,1}, p_{SD,1}, ..., p_{SD,i}, ..., p_{SD,n}\}$$

the inter-packet delay (IPD) between any two packets can be calculated as $\tau_{SD,i,i+1} = t_{SD,i+1} - t_{SD,i}$ where $t_{SD,i}$ is the timestamp of the packet $p_{SD,i}$. Without loss of generality, for the ease of illustration, we will remove "SD" from the notation and denote the packet stream P_{SD} as P and IPD $\tau_{SD,i,i+1}$ as τ_i.

The encoder selects $2m$ packets $\{p_{r_1}, p_{r_2}, ..., p_{r_{2m}}\}$ out of the n packets of packet stream P. The selected packets are paired with another $2m$ packets (outside of the initially selected $2m$ packets) to create $2m$ pairs such that each pair is constructed as $\{p_{r_z}, p_{r_z+x}\}$ where $x \geq 1$ and $z = 1, ..., 2m$. Therefore, it is assumed that the packet stream has at least $4m$ packets. The IPD between each pair of packets can be calculated as;

$$\tau_{r_z} = t_{r_z+x} - t_{r_z} \tag{2}$$

Given that the $2m$ packets are selected independently and randomly, we model the IPDs as *independently and identically distributed (IID)* random variables with a common distribution. The IPD values are then divided into 2 groups. Since we had $2m$ pairs of packets, each group will have m IPD values. Let the IPD values of the two groups be denoted by τ_k^1 and τ_k^2 ($k = 1, ..., m$), respectively. It follows that both τ_k^1 and τ_k^2 are IID. Therefore, the expected values μ (and the variances) of the two distributions are equal. Let Δ be the average difference between the two IPD distributions:

$$\Delta = \frac{1}{m} \cdot \sum_{k=1}^{m} \frac{\tau_k^1 - \tau_k^2}{2} \tag{3}$$

Then, we can calculate the expected value and variance of Δ:

$$\mathbb{E}[\Delta] = \mathbb{E}\left[\tau_k^1\right] - \mathbb{E}\left[\tau_k^2\right] = 0\ , \qquad \mathrm{Var}(\Delta) = \frac{\sigma^2}{m}\ .$$

Where σ^2 is the variance of the distribution $\frac{\tau_k^1 - \tau_k^2}{2}$. In other words, the distribution of Δ is symmetric and centered around zero. The parameter m is referred to as the *sample size*.

The core idea of our watermarking approach is to intentionally delay a selected set of packets to shift the Δ distribution left or right to encode the watermark bits in the timing information of the packets. Specifically, the distribution of Δ can be shifted along the x-axis to be centered on $-\alpha$ or α by decreasing or increasing Δ by α, where α is called the *shift amount*. As a result, the probability of Δ being negative or positive will increase. Concretely, to embed bit 0, we decrease Δ by α. To embed bit 1, we increase Δ by α. Decreasing Δ can be done by decreasing each $\frac{\tau_k^1 - \tau_k^2}{2}$ by α (Equation 3). Decreasing $\frac{\tau_k^1 - \tau_k^2}{2}$ can be achieved by decreasing each τ_k^1 by α and increasing each τ_k^2 by α. It is easy to see that increasing Δ can be done in a similar way. Decreasing or increasing one IPD (τ_k^1) is achieved by delaying the first packet or the second packet of the pair, respectively.

The encoded watermark can be detected by calculating Δ and checking if Δ is positive or negative. If $\Delta > 0$, bit 1 is decoded. Otherwise (if $\Delta \leq 0$) bit 0 is decoded. This scheme can be extended to a w-bit watermark (ω_{SD}) by repeating the above process w times. During the decoding process, a w-bit watermark (ω'_{SD}) is extracted from the packet stream and if the hamming distance between ω_{SD} and ω'_{SD} is lower than a predefined *error margin* δ, we can conclude that the watermark embedded at the source S is detected at the receiver. If the watermark does not match, an attack is flagged.

Figure 5 shows the distribution of Δ and the corresponding distribution after shifting it by $\alpha > 0$. Since our scheme is probabilistic, there is a probability that the embedded watermark bits will be incorrectly decoded, thus leading to false alarms (false positives) or missed detection (false negatives). This is because for any $\alpha > 0$, a small portion of the distribution of Δ falls outside the range $(-\infty, \alpha]$. Therefore, if we embed bit 0, there is a small probability that the bit will be incorrectly decoded as 1. It can be seen that this probability is the same as the probability that a sample from the unshifted distribution takes a value outside the range $(-\infty, \alpha]$. Similarly, a bit encoded to be 1 can be decoded incorrectly because samples from Δ have a small probability of falling outside the range $[-\alpha, \infty)$. However, we can tune parameters m (sample size), α (shift amount) and δ (error margin) to achieve a very high (nearly 100%) decoding success rate (Section VII).

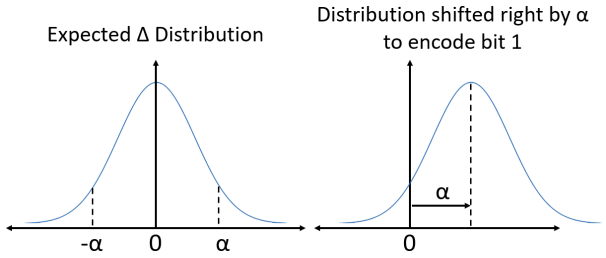


Fig. 5: Example showing the Δ distribution shifted by α.

To provide formal guarantees, we define the *bit decoding success rate* (BDSR) as the probability of the embedded watermark bit being decoded correctly (for a shift amount of α). We denote this quantity by $\Pr[\Delta < \alpha]$. Note that the BDSR also depends on m and σ^2, but this is not explicit in the notation $\Pr[\Delta < \alpha]$ because it is implicitly captured by Δ. We give an illustrative example to further explain this concept.

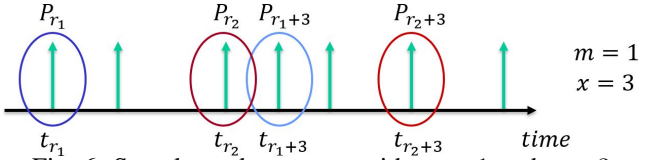


Fig. 6: Sample packet stream with $m = 1$ and $x = 3$.

Illustrative Example: Figure 6 shows a sample packet stream in the time domain with packet injection times. For ease of explanation in this example, m is set to one and therefore, two packets ($2m$) are selected from the packet stream (P_{r_1} and P_{r_2}). Both packets are paired with two other packets that are x (=3) packets away in the packet stream (P_{r_1} with P_{r_1+3} and P_{r_2} with P_{r_2+3}). The IPD between each pair is calculated as $\tau_{r_1} = t_{r_1+3} - t_{r_1}$ and $\tau_{r_2} = t_{r_2+3} - t_{r_2}$. The two IPD values are then divided into two groups and Δ calculated according to Equation 3 as $\frac{\tau_{r_1} - \tau_{r_2}}{2}$ (sum for all m and division by m not shown since $m = 1$). We repeated the process using a packet stream that had more than 3000 packets obtained by running a simulation using the gem5 architectural simulator [44] on a

real benchmark. An 8×8 Mesh NoC was modelled using the Garnet2.0 [49] interconnection network model. The node in the top left corner (node S) ran the RADIX benchmark from the SPLASH-2 benchmark suite [43]. One memory controller was modelled and attached to the node in the bottom right corner (node D) so that the memory requests always traverse from S to D. Figure 7 shows the histogram collected at the NI of S for the distribution of Δ with $m = 1$ and $x = 3$. Packets were collected at random with the above parameter values to plot Δ. We can observe from Figure 7 that the distribution closely approximates the distribution we expected. The calculated sample mean ($\mathbb{E}[\Delta]$) for this particular example was 0.0053, which is very close to zero. Increasing the number of selected packets ($2m$) further increases the likelihood of the sample mean being zero.

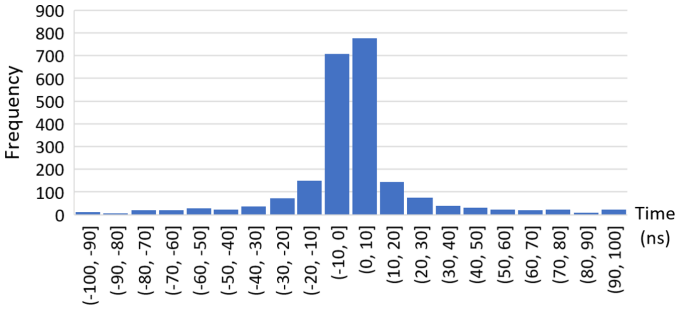


Fig. 7: Distribution of Δ with $m = 1$ and $x = 3$.

D. Watermark Encoder and Decoder

As outlined in Section V-B, our watermarking scheme includes a shared secret between S and D, which is "hard" for any other node to guess or deduce. In addition, several parameters are shared between S and D. Specifically, S and D share the tuple $\langle m, \alpha, w_{SD}, \mathbb{K} \rangle$. The first three parameters were introduced in Section V-B as the sample size (m), the shift amount (α), and the unique watermark that represents P_{SD} (w_{SD}). The length of w_{SD} (w) can be derived from w_{SD}. In addition, $\mathbb{K}$ is a secret which is used to derive a key for the encryption scheme and a seed $\mathbb{S}$ using a key derivation function. $\mathbb{S}$ is used to seed the pseudo-random number generator which selects the $2m$ IPDs. We assume the attacker does not know w_{SD} or $\mathbb{K}$, but may know m and α.

1) Watermark Encoding Process: When the watermark encoder, which is integrated in the NI of node S, receives packets from its local IP with the destination node D, it encodes the watermark according to the process outlined in Section V-C and the shared secret between S and D. The selection of the IPDs that construct the Δ distribution needs to be deterministic so that the process is identical for the watermark encoder and decoder, and it needs to ensure that an attacker cannot replicate the same behavior. To achieve this, we need a method to pair packets deterministically based on the shared secret, but that appears uniformly random to the attacker (who does not know the shared secret). We propose to implement this using a pseudo-random number generator (PRNG) seeded (i.e., initialized) with $\mathbb{S}$ (or something derived from it). This ensures that the encoder and decoder produce the *same* sequence of random numbers. Further, an attacker (who does not know the seed) cannot predict the next PRNG output, even with the knowledge of the previous output [50]. There are many possible hardware implementations of PRNG that are suitable for our framework [51]. For example, our approach can be built on top of LFSR-based PRNGs where the period (repeat sequence) can be determined and optimized.

Let $\mathcal{F}$ denote the selection function that given a packet stream, selects and divides $2m$ IPDs into two groups, each of size m. We choose a window of packets and pair two random packets together from each window. Therefore, to construct $2m$ IPDs, $2m$ such packet windows are required. The operation of $\mathcal{F}$ used in our method is outlined in Algorithm 1. The PRNG seeded with $\mathbb{S}$ is used to randomly generate two integers r_z and x (line 1) such that $0 \leq r_z \leq W-1$ and $0 < x$ and $r_z + x \leq W - 1$, where W is the size of the window. This can be done using rejection sampling to ensure that $r_z \neq x$ and then calling the smaller integer r_z and the larger $r_z + x$. The packet at the index r_z (p_{r_z}) is paired with the packet that is x packets away giving the random pair $\{p_{r_z}, p_{r_z+x}\}$ (lines 4-5). The calculated IPD values are then evenly divided into two groups (lines 6-11).

Since $2m$ IPDs are required to encode a 1-bit watermark, w iterations of the procedure $\mathcal{F}$ are required to encode the w-bit watermark. When encoding one watermark bit, the distribution discussed in Section V-C holds only when each pair of packets is the same distance x apart from each other. Therefore, the same r_z and x values are used for each iteration of k. When encoding another watermark bit, another iteration of $\mathcal{F}$ is required in which another pair of r_z and x values will be generated by the PRNG. To ensure that the same r_z and x values are not generated for subsequent watermark bits, the PRNG must be seeded only once. An example to show how the selection function can be used to encode a w-bit watermark including how to select the window is given in Section VII.

Algorithm 1 - Selection Function $\mathcal{F}$

Input: Seed $\mathbb{S}$
Output: Two IPD groups used to encode one watermark bit
Procedure: $\mathcal{F}$

1: $r_z, x \leftarrow PRNG(\mathbb{S})$
2: **for all** $k = 1, ..., 2m$ **do**
3: $\quad A \leftarrow \text{selectNextWindow}(P_{SD})$
4: $\quad p_{r_z} \leftarrow A[r_z]$
5: $\quad p_{r_z+x} \leftarrow A[r_z + x]$
6: $\quad \tau_{r_z} \leftarrow t_{r_z+x} - t_{r_z}$
7: $\quad$ **if** k is odd **then**
8: $\quad\quad \tau^1_k \leftarrow \tau_{r_z}$
9: $\quad$ **else**
10: $\quad\quad \tau^2_k \leftarrow \tau_{r_z}$
11: **return** $[\{\tau^1_1, \tau^1_2, ..., \tau^1_m\}, \{\tau^2_1, \tau^2_2, ..., \tau^2_m\}]$

2) Watermark Decoding Process: Node D upon examining the packet stream P_{SD}, decodes the w-bit watermark w'_{SD} by following the process outlined in Section V-C and the shared secret tuple. The decoder concludes that the watermark is valid if the Hamming distance between w_{SD} (taken from the shared secret tuple) and w'_{SD} (decoded from the received packet stream P_{SD}) is less than or equal to the error margin

δ. Formally, the watermark is valid if;

$$\mathcal{D}(w_{SD}, w'_{SD}) \leq \delta \tag{4}$$

where $\mathcal{D}$ is the Hamming distance between two bit strings and $0 \leq \delta \leq w$. The reason for allowing an error margin δ and not looking for an exact match is that no matter how large the shift amount α is, there is a probability that the watermark is decoded incorrectly as discussed in Section VI-A. Tuning parameter δ allows us to minimize this probability. As shown in Section VI-B, it allows to minimize the impact of the attack.

E. Managing Shared Secrets

The watermark encoder and decoder operation introduced in Section V-D relies on shared secret tuples between nodes to make sure the watermarking scheme cannot be compromised. To facilitate this, an efficient way to generate and manage such secrets is required. Developing an efficient management mechanism is beyond the scope of this work and many previous studies have addressed this problem in several ways. One such example is the key management system proposed by Lebiednik et al. [28]. In their work, a separate IP called the *key distribution center* (KDC) handles the distribution of keys. Each node in the network negotiates a new key with the KDC using a pre-shared portion of memory that is known by only the KDC and the corresponding node. The node then communicates with the KDC using this unique key whenever it wants to obtain a new key. The KDC can then allocate keys and inform other nodes as required. AE schemes also rely on the services of a KDC to manage shared keys between nodes [16], [13], [7], [11]. Our proposed digital watermarking scheme can be integrated with a similar key generation and management mechanism.

VI. Theoretical Analysis

In this section, we provide some mathematical guarantees about the correctness and security of the watermarking scheme which we further validate with experimental results in Section VII. First, we provide a bound on BDSR during normal operation (Section VI-A). Then we evaluate the impact of an attacker on BDSR (Section VI-B). Finally, we present how the error margin δ can be selected such that it maximizes the chance of successfully decoding the watermark while minimizing the chances of an attack if the attacker is aware of our detection method (Section VI-C).

A. Bit Decoding Success Rate During Normal Operation

Given this watermark encoding/decoding scheme, it is clear that larger the shift amount α is, the higher the bit decoding success rate (BDSR) will be. However, having arbitrarily large α is not feasible in systems with real-time constraints. In this section, we show that we can achieve close to 100% BDSR for arbitrarily small α by changing the sample size m.

As discussed in Section V-C, a watermark bit can be decoded incorrectly if at the receiver's end, $|\Delta| > \alpha$. Therefore, we should analyze the behavior of $\Pr[|\Delta| > \alpha]$. There are several well-established statistical tools for this, but in particular we can use concentration results, also known as tail bounds. Since the IPDs are bounded and independent, we can use Hoeffding's inequality (introduced in Section V-A1) and equations from Section V-C related to the distribution of Δ;

$$\Pr[|\Delta| \geq \alpha] \leq e^{\left(-\frac{m\alpha^2}{2\sigma^2}\right)} \tag{5}$$

$$\text{Using symmetry;} \qquad \Pr[\Delta < \alpha] \geq 1 - \frac{1}{2}e^{\left(-\frac{m\alpha^2}{2\sigma^2}\right)} \tag{6}$$

Therefore, we can observe that the BDSR is lower bounded by a value that depends on α and m. The results show that irrespective of the distribution of the IPDs, for arbitrarily small α values, we can always take the BDSR close to 100% by increasing the sample size m. In other words, no matter how small the shift amount α needs to be to abide by the timing constraints of the system, we can still achieve high BDSR by selecting more packets in each IPD group.

B. Impact of an Attack on the Bit Decoding Success Rate

Having established mathematical guarantees about BDSR during normal operation, we shift our focus to explore how BDSR of legitimate packet streams can be affected by an attack. According to the threat model, the Trojan infected router copies packets and sends them to a malicious application running on a different IP. As a result, more packets are introduced to the network which can cause congestion. All packets in the network can be delayed because of this. Therefore, the attack can introduce additional delays to the legitimate packet streams. It is safe to assume that these additional delays are finite. If the attacker delays packets indefinitely through congestion, the attack is no longer an eavesdropping attack, but rather a flooding type of denial-of-service attack [4] that is beyond the scope of this paper.

Given that the Trojan-infected router does not know which packets were selected by the watermark encoder (as explained in Section V-D), the delay introduced by the attacker (whatever it is) on the selected IPDs is IID from the perspective of S and D. Using this insight, we can analyze Δ', which is the distribution after modifying Δ defined in Equation 3 with the added delays, and conclude that;

$$\Pr[\Delta' < \alpha] \geq 1 - \frac{1}{2}e^{\left(-\frac{m\alpha^2}{2(\sigma+\sigma_d)^2}\right)} \tag{7}$$

where σ_d is the added delay variance due to the added congestion. Observe that the only change is the increase in variance caused by the attacker. We can choose σ_d depending on the amount of congestion the attacker is willing to cause without risking being detected. Similar to the argument we made when reasoning about the BDSR using Equation 6, we can see that BDSR is lower bounded and by manipulating the sample size, we can make the BDSR arbitrarily close to 100%. Therefore, the impact on the watermarking detection is a bounded increase of variance on an otherwise 100% successful watermarking scheme. As the illustrative example that calculates BDSR in Section VII-B outlines, the success rate can be brought very close to 100% even with the selection of a modest value for m.

C. Optimal Error Margin Selection

As discussed in Section V-D, the use of the error margin δ instead of an exact match between the decoded and the expected watermark, allows us to tune δ to maximize the *watermark detection success rate* (WDSR). Unlike BDSR, which refers to the success of decoding a single bit, WDSR considers the entire watermark with w bits. The probabilistic nature of our watermarking scheme leaves a small probability that the watermark will be incorrectly decoded irrespective of the values chosen for the parameters. While this probability is small, efficient selection of δ can push WDSR as close as possible to 100%. On the other hand, using a larger error margin also increases the success of potential attacks. Indeed, assuming that the attacker is aware of our detection strategy, the best strategy for an attacker to eavesdrop on data without being detected is to try to *forge* a watermark. If he succeeds, then the duplicated packets will be accepted as valid by the node that runs the accomplice application and our proposed watermarking-based defense will be defeated. We call the success probability of such a forging attack the *watermark forging success probability* (WFSP). The goal of the detection scheme is thus to set the parameters such that WDSR is maximized while minimizing WFSP. We explore how this can be achieved in this section.

1) Maximizing Watermark Detection Rate: The probability of incorrectly decoding a bit was formalized using the metric BDSR as $\Pr[\Delta < \alpha]$. Considering symmetry, let $\vartheta = \Pr[-\infty < \Delta < \alpha] = \Pr[-\alpha < \Delta < \infty]$. Then for a w-bit watermark, probability of accurately decoding all w bits will be ϑ^w. Therefore, the expected WDSR can be calculated as;

$$\sum_{i=0}^{\delta} \binom{w}{i} \vartheta^{w-i}(1-\vartheta)^i \tag{8}$$

We can see that with a large δ, the expected WDSR increases. We observe from Equation 8 that;

$$\sum_{i=0}^{\delta} \binom{w}{i} \vartheta^{w-i}(1-\vartheta)^i \geq \vartheta^w \tag{9}$$

Therefore, we can make the expected WDSR larger than the desired WDSR by increasing ϑ. Revisiting Equation 7, we observe that ϑ can be made sufficiently close to 1 by increasing the sample size m irrespective of α, σ and σ_d. Therefore, we can conclude that in theory, it is possible to make WDSR close to 100% even with a modest error margin.

2) Minimizing Risk of Watermark Forging Attacks: While increasing δ can increase WDSR, larger the δ, larger the expected WFSP will be. We address this in two steps. First, we select watermarks such that under a given error margin δ, the probability that one watermark can be incorrectly decoded as another watermark (watermark collision) is minimized. Then, we discuss the case where an attacker, after knowing our detection mechanism, tries to inject duplicated packets such that the decoder at the receiver incorrectly validates the watermark (watermark forging) and accepts the duplicated packet steam as valid.

The problem of selecting distinct w-bit watermarks for each source-destination pair can be recast as the problem of selecting distinct codewords. This is a well-established problem that has been extensively studied in the information theory literature. Indeed, it is known that for any given set of distinct codewords, if the minimum Hamming distance between any two codewords is at least $2\delta + 1$, a nearest neighbor decoder will always decode correctly when there are δ or fewer errors [52]. Therefore, if the watermarks are chosen such that any two watermarks are at least $2\delta+1$ distance apart, the probability of a watermark collision is minimal. We select the number of bits in the watermark w such that this property is satisfied using the method explained in Section V-A2. An example of how w is selected is given in Section VII-B2.

Even if w is selected such that watermark collision probability is minimized, an attacker may still try to impersonate a legitimate sender. Assume that w_{SD} and w_{SY} are valid watermarks with distance $2\delta + 1$ (minimum possible distance between two watermarks) between nodes S and D and S and Y, respectively. A Trojan-infected router in the path from S to D duplicates packets and sends to an accomplice application in node Y. For Y to accept the duplicated packet stream as a legitimate packet stream coming from S, the watermark of the duplicated packet stream should match w_{SY}. We refer to this attack as a *watermark forging* attack.

Section V-D and Section V-E detailed how watermarks are kept unknown to any other parties, except for the sender and receiver in a packet stream, using shared secrets. Therefore, the attacker's method to forge a watermark can be reduced to a random bit flipping game with the goal of matching w_{SY}. Random bit flipping is achieved by randomly delaying the duplicated packets in P_{SD}. For the attacker to win the game, w_{SD} should change to w_{SY}. Since the minimum distance between any two watermarks is $2\delta + 1$, considering the error margin of δ, the minimum required number of bit flips is $\delta+1$. Therefore, the attacker should flip at least $\delta + 1$ bits to win the game. However, flipping the wrong bits can take the target even further. Therefore, the best chance for the attacker to win the game is if it flips the correct $\delta + 1$ bits of w_{SD} to match w_{SY} (to end up within the error-margin of w_{SY}, i.e., within δ-Hamming distance of w_{SY}). The probability that the attacker flips the correct $\delta + 1$ bits at any given round of the game is thus: $\binom{w}{\delta+1}^{-1}$. Assuming the attacker plays n times, the attacker's probability of winning, or in other words, the probability of successfully forging the watermark (WFSP) at least once (after n attempts) is;

$$1 - \left[1 - \frac{1}{\binom{w}{\delta+1}}\right]^n \tag{10}$$

Observe that by manipulating w and δ, this probability can be made arbitrarily small. Furthermore, n cannot be arbitrarily large because if the probability of winning in the first few attempts is low, then the attacker will be detected before the attacker can successfully forge the watermark.

This allows us to conclude that we can make WDSR close to 100% and WFSP close to 0%. Equations 7, 8 and 10 combined give us the theoretical trade-off model between WDSR and WFSP. However, we cannot accommodate arbitrarily large m and w in practical scenarios. Therefore, in the

next section, we perform experimental evaluation and discuss realistic values that can be achieved under our threat model and architecture.

VII. Experimental Results

In this section, we experimentally evaluate the theoretical models established in previous sections and choose the parameters that give the optimum results. The selected parameters are then used to explore the performance gain achieved by using our method compared to traditional AE based schemes.

A. Experimental Setup

We evaluated our approach by modeling an NoC-based SoC using the cycle-accurate full-system simulator - gem5 [44]. "GARNET2.0" interconnection network model that is integrated with gem5 was used to model an 8x8 Mesh 2D NoC [49]. To ensure the accuracy of our simulator model when compared to real hardware, we used the simulator framework proposed in [53], which has validated simulator results with results from the Intel Knights Landing (KNL) architecture (Xeon Phi 7210 hardware platform [54]), when setting up the experimental environment. Figure 8 shows an overview of the NoC-based SoC model. Each IP was modeled as a processor core executing a given task at 1GHz with a private L1 Cache. Eight memory controllers were modeled and attached to the IPs in the boundary providing the interface to off-chip memory. In case of a cache miss, the memory request/response messages were sent to/from memory controllers as NoC packets. The NoC was modeled with 3-stage (buffer write, route compute + virtual channel allocation + switch allocation, and link traversal) pipelined routers with wormhole switching and 4 virtual channel buffers at each input port. Packets are routed using the deadlock and livelock free, hop-by-hop, turn-based XY deterministic routing protocol.

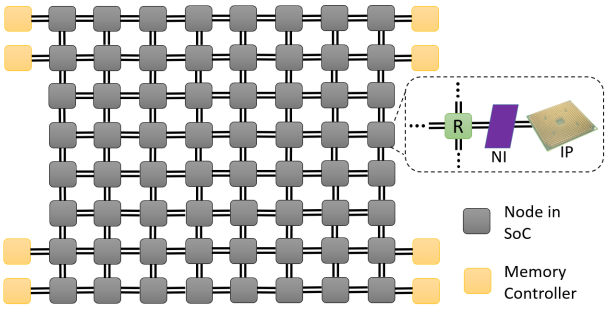


Fig. 8: 8x8 Mesh NoC setup used to generate results.

Each processor core in the SoC was assigned an instance out of FFT, RADIX (RDX), FFM and LU benchmarks from the SPLASH-2 benchmark suite [43]. Each simulation round can in theory, give $\binom{64}{2} \times 2 = 4032$ packet streams (assuming two-way communication between any pair out of the 64 nodes) and the number of iterations that depended on the number of benchmarks (four in our case) can give $4 \times \binom{64}{2} \times 2 = 16{,}128$ packet streams. However, depending on the address mapping, only some node pairs out of all the possible node pairs communicate. Our simulations generated 3072 packet streams for all benchmarks between 1024 unique node pairs which we used to evaluate our method. However, to decide the number of bits in the watermark w, looking at only the number of unique node pairs is not sufficient because to avoid watermark collisions, the Hamming distance between any two watermarks should be at least $2\delta + 1$. According to Section V-A2, as δ increases, w increases as well. Therefore, more packets are required to encode the watermark and as a result, the time to detect an ongoing attack increases (more packets need to be observed before recognizing the watermark). Increasing m has a similar impact. Increasing α increases the application execution time and it takes longer to detect eavesdropping attacks. This motivates us to explore optimum parameter (m, α and δ) values such that WDSR is maximized and attack detection time, execution time as well as WFSP are minimized.

B. Parameter Tuning

We first explore m and α when encoding a single watermark bit and then extend the discussion to consider WDSR, WFSP, execution time and detection time.

1) Bit Decoding Success Rate Behavior with m and α: When embedding one watermark bit in a packet stream, Equation 6 gives a theoretical estimate of the BDSR. To compare the theoretically expected BDSR with experimental results, we use a non-overlapping sliding window of λ packets and select $2m$ IPDs according to the method in Section V-D1. One bit is encoded in each of the 3072 selected packet steams following the same methodology and decoded at the receiver's side according to the method introduced in Section V-C. $\lambda = 8$ is chosen to ensure adequate randomness in the IPD selection process. A detailed analysis of λ value selection is given in Section VIII. We keep $\alpha = 60$ns fixed and vary m from 2 to 15. Results are shown in Figure 9. We compare the outcome from our experiments with the theoretical model (Equation 6). For example, expected BDSR for $m = 4$, $\alpha = 60$ns and $\sigma^2 = 2662$ is calculated as;

$$\Pr\left[\Delta < 60\right] \geq 1 - \frac{1}{2} e^{\left(-\frac{4 \times 60^2}{2 \times 2662}\right)} \approx 0.967$$

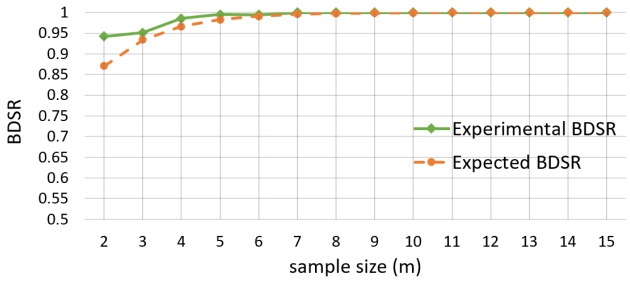


Fig. 9: BDSR variation with sample size m. $\alpha = 60ns$.

We now fix $m = 4$ and vary α from 10ns to 100ns to explore BDSR variation with α. Figure 10 shows the comparison between the theoretical model (Equation 6) and results generated from our experiments. The experimental results in both Figure 9 and Figure 10 show that our theoretical model gives an accurate bound on BDSR. As α and m are increased, BDSR converges to 1. However, our goal is to detect any attack with high accuracy while incurring minimum performance overhead. Therefore, BDSR is not the

only deciding factor. As α and m is increased, the execution time of the application/benchmark running with our attack detection mechanism increases as well. α and m should be chosen such that this trade-off is maintained.

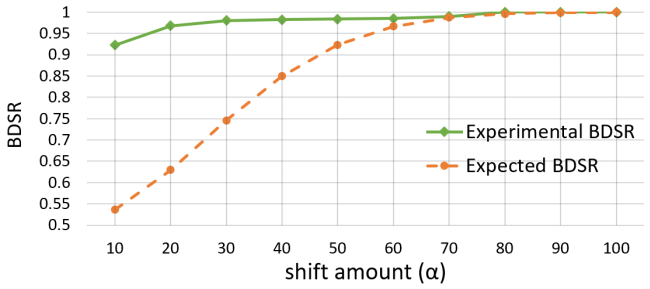


Fig. 10: BDSR variation with shift amount α. $m = 4$.

While Figure 9 and Figure 10 show how BDSR varies with m and α, both figures had one parameter fixed while varying the other. To observe how both m and α effect the BDSR as well as the execution time, we did a grid search in the ranges $2 \leq m \leq 10$, $10 \leq \alpha \leq 80$ and $w = 20$ and eliminated cases where expected BDSR was less than 0.95 and execution time increase was more than 5%. These thresholds were chosen to achieve the optimum balance in the trade-off. Results are shown in Figure 11. $w = 20$ is chosen because, to provide a unique watermark for each communicating node pair (1024 in our experiments), 10 bits are required. 10 additional bits are kept to allow error margins as well as to avoid collisions. However, as discussed in Section VII-B2, w can be further optimized leading to a better execution time. Execution time increase is measured as the average execution time increase as a percentage when benchmarks are run with our approach compared to Default-NoC introduced in Section IV. Out of the possible combinations in Figure 11, we pick $m = 4$ and $\alpha = 60$ as it gives an adequate trade-off for our exploration.

α \ m	2	3	4	5	6
50	X	X	X	0.952 0.989 4.27%	0.970 0.991 4.75%
60	X	X	0.967 0.985 4.17%	0.983 0.995 4.73%	X
70	X	0.968 0.971 3.89%	0.987 0.990 4.54%	X	X
80	0.955 0.960 3.42%	0.986 0.989 4.19%	X	X	X

Fig. 11: BDSR and execution time variation with m and α. w fixed at 20. The green cells show expected BDSR, purple show experimental BDSR and yellow indicate execution time increase. Crosses indicate either expected BDSR or execution time increase falling beyond our selected thresholds.

2) Choosing δ and w: With the values selected for m and α, we explore the impact of the error margin δ on WDSR. To calculate expected WDSR according to Equation 8, w should be decided. However, the value of w is dependant on the value we select for δ. Therefore, we explore the behavior of expected WDSR with respect to δ for several fixed w values ($w \in \{14, 16, 18, 20\}$). Results are shown in Figure 12. $\delta = 0$ represents exact matches between the decoded watermark and the expected watermark without using an error margin. The importance of using δ is evident when the scenario of looking for exact matches ($\delta = 0$) is compared with any other δ value. For example, for the values $\vartheta = 0.967$ and $w = 20$, WDSR with exact matches is $\vartheta^w = 51.1\%$ whereas for the same ϑ and w values with an error margin of 2, WDSR is 97.3%.

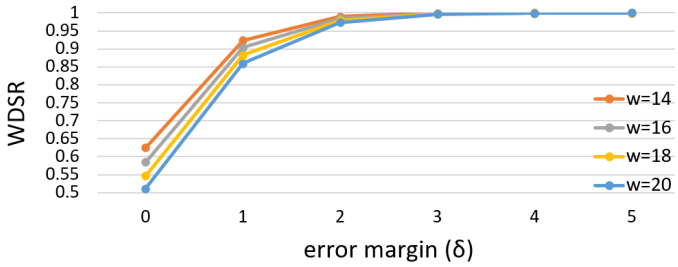


Fig. 12: Expected WDSR variation with error margin δ for several w values. m and α fixed at 4 and 60ns, respectively.

As outlined in Section VI-C2, the chosen δ value affects the chances of the attacker succeeding in a forging attack (WFSP). To evaluate the impact, we explored WDSR (Equation 8) and WFSP (Equation 10) values for different combinations of w and δ. However not all w and δ values can co-exist if watermark collisions are to be avoided. Assume that the chosen δ value is 2. As outlined in Section VI-C2, for two watermarks not to collide, they should be at least $2\delta + 1$(=5 if $\delta = 2$) Hamming distance apart. Since there are 1024 unique node pairs, we can set w as the minimum number of bits required to generate 1024 unique codewords such that the minimum Hamming distance between any two codewords is 5. In other words, we are looking for w such that $A(w, 5) \geq 1024$ according to Section V-A2. From [48], we can derive $w \geq 18$. Therefore, to ensure that there are no collisions between watermarks with an error margin of 2, at least 18 bits are required for the watermark. Similarly, we can derive $w \geq 21$, for $\delta = 3$, and $w \geq 14$ for $\delta = 1$. Since increasing w has an impact on execution time as well, for each δ value, we pick the two smallest possible w value such that there are no watermark collisions. Table II shows expected WDSR, WFSP values, experimental WDSR value and execution time increase for the selected configurations.

TABLE II: WDSR, WFSP and execution time increase for varying w and δ. $\vartheta = 0.967$, $n = 10$.

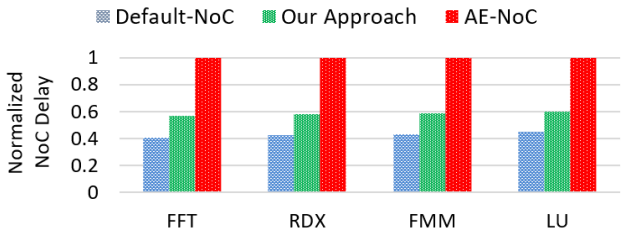
δ	w	Expected WDSR	WFSP	Experimental WDSR	Execution Time Increase
1	14	0.9238	0.1046	0.9538	3.49%
1	15	0.9139	0.0912	0.9512	3.61%
2	18	0.9797	0.0121	0.9801	3.95%
2	19	0.9765	0.0102	0.97884	4.06%
3	21	0.9955	0.0075	0.9987	4.29%
3	22	0.9946	0.0064	0.9964	4.40%

These results strongly support our claim that WFSP can be made arbitrarily small by manipulating w and δ. We observe

from Figure 12 that WDSR converges to 1 starting $\delta = 2$. Furthermore, observing values in Table II, we can pick $\delta = 2$ and $w = 18$ as a configuration that gives an adequate trade-off. The chosen values gives an experimental WDSR of 98%. It is important to note that the presence of watermark or detection mismatch does not affect the functional behavior since the watermark is embedded in the timing information, not in the packet content. In terms of attack detection, our approach is 98% accurate for the chosen parameters. The remaining 2% consists of both false positive and false negative cases.

C. Performance Evaluation

With the selected parameters, $m = 4$, $\alpha = 60$, $\delta = 2$, $w = 18$, we explore the performance improvement achieved by our method compared to the traditional AE based defenses. Section IV introduced two scenarios - *Default-NoC* and *AE-NoC* against which we evaluate the performance of our approach (digital watermarking-based attack detection coupled with encryption). As outlined in Section II, AE-NoC is selected for comparison since it is the most widely adopted approach to mitigate similar threats according to existing literature. NoC delay and execution time comparison are shown in Figure 13 considering Default-NoC, AE-NoC and our watermarking based attack detection method. Our approach only increases the NoC delay by 27.9% (26.3% on average) and execution time by 5.2% (3.95% on average) compared to the default NoC whereas AE-NoC increased NoC delay by 59% (57% on average) and execution time by 17% (13% on average). Therefore, our method has the ability to significantly improve performance compared to other state-of-the-art security mechanisms intended at preventing eavesdropping attacks.



(a) NoC delay

(b) Execution time

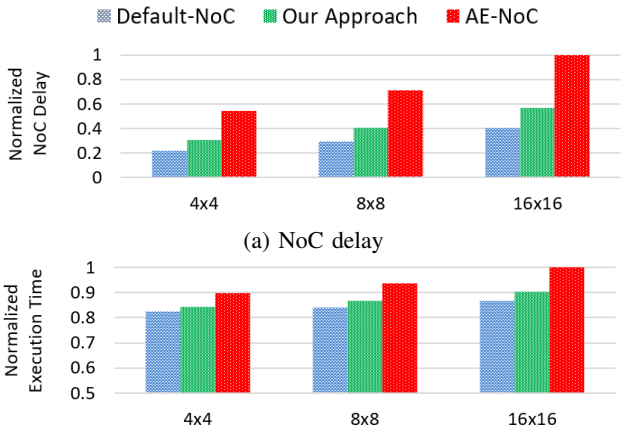
Fig. 13: NoC delay and execution time comparison.

In addition to execution time comparison, time taken to detect an ongoing attack (detection time) is also critical. Detection time is calculated as the time taken to decode the complete watermark from a packet stream. As soon as the w-bit watermark is decoded and validated, any eavesdropping attack can be detected. Table III shows detection time for each benchmark normalized to total execution time. This shows that our watermark detection scheme is capable of detecting any eavesdropping attacks in a timely manner.

TABLE III: Attack detection time for different applications/benchmarks. Each value is normalized to the corresponding benchmark execution time.

FFT	RDX	FMM	LU
6.56E-3	4.8E-5	1.9E-4	3.9E-4

To evaluate the scalability of our approach, we ran the same experiments using the FFT benchmark on 4x4, 8x8 and 16x6 Mesh NoCs. Results are normalized to the highest runtime and shown in Figure 14. The increase in the number of nodes from 4x4 to 8x8 introduced a 9.6% increase in NoC delay and 2.1% in execution time. When the number of nodes were increased from 8x8 to 16x6, the NoC delay increased by 16.2% and as a result, execution time increased by 3.7%. However, this increase is similarly applicable to all three scenarios (not only our approach) as seen in Figure 14, since the NoC size increase causes similar packet transfer delays in other scenarios as well.



(a) NoC delay

(b) Execution time

Fig. 14: Performance comparison across NoC configurations.

In summary, these results validate our theoretical model and provide a framework to tune the parameters such that eavesdropping attacks can be detected quickly with high accuracy while providing a significant performance improvement compared to existing state-of-the-art solutions.

VIII. Discussion

The security of the watermarking scheme depends on the secrecy of some parameters (Section V-D). Parameters include the watermark w_{SD} as well as the key $\mathbb{K}$ for each P_{SD}. A key distribution center (KDC) acts as a trusted dealer to distribute these parameters. In this section, we discuss security implications if some of these assumptions do not hold.

A. Eliminating the Trusted Dealer

In the absence of a trusted dealer, each communicating node pair will have to agree on a watermark and a key. While this can be facilitated by key-exchange protocols such as the Diffie-Hellman key exchange, the lack of a trusted dealer can cause duplicated watermarks (watermark collisions). If watermarks are selected uniformly at random to minimize the chances of collision, according to the birthday bound, the number of bits assigned to the watermark should be double of what is required. For example, if an 18-bit watermark is required in the presence of a trusted dealer, 36 bits are

required in its absence because of the birthday bound. While our watermarking scheme can give better accuracy and less collisions for a 36-bit watermark, the execution time as well as the detection time will increase. Therefore, a designer needs to carefully select the size of the watermark to minimize the collision without violating the performance budget.

B. What Can Be Inferred from Packet Timing?

It is important to note that the watermark is encoded in the IPD values, not in the individual packet injection/received times. Furthermore, packet injection times can vary depending on the behavior of the application as well. There can be phases in the application execution where more packets are injected to the NoC whereas in some other phases, delay between packet injections is comparatively high. Therefore, "guessing" the watermark cannot be easily accomplished by merely observing packet arrival times. Moreover, the only way for an attacker to forge the watermark successfully is to know both the watermark and the PRNG seed.

Indeed, even if the watermark could be inferred from packet timing, the PRNG seed cannot be inferred from packet timing information due to cryptographic guarantees of using a PRNG. In the next section, we assume that the watermark is known by the adversary but not the PRNG seed and analyze the probability that an attacker can forge the watermark. This probability can be reduced to a random bit flipping game (probability = $\frac{1}{2}$).

C. Watermark Is Not a Secret Anymore?

Assume that the attacker knows the watermark, but not the PRNG seed. To forge the watermark, the attacker must select the two correct packets (that forms the IPD) from each window. Observe that without the PRNG seed, the attacker's probability of correctly guessing the two packets from a given window is $1/\binom{\lambda}{2}$ (Case I). Similarly, we can derive that the probability of two packets chosen by the attacker partially overlapping with the correct two packets and the probability of the attacker not selecting either one of the two correct packets are $2(\lambda-2)/\binom{\lambda}{2}$ (Case II) and $\binom{\lambda-2}{2}/\binom{\lambda}{2}$ (Case III), respectively. Therefore, the higher the value chosen for λ, the lower the chances of a successful attack. The probability of the attacker *not* selecting either one of the two packets correctly (Case III) goes above 0.5 at $\lambda = 8$. In the overlapping scenario, if the first packet selected by the attacker is the correct second packet (or vice versa), delaying it will give the incorrect watermark bit. However, to give a conservative estimate, we ignore that possibility and use $\lambda = 8$ so that the probability of selecting both packets incorrectly is at least $\frac{1}{2}$. This analysis shows that our watermarking scheme can be tuned to work even in scenarios with very strong security assumptions such as the watermark being leaked to the attacker. Additionally, for systems which require even stronger security, another layer of security can be added if we rotate the watermark assigned between each pair of nodes after some number of iterations.

IX. Conclusion

In this paper, we introduced a lightweight eavesdropping attack detection mechanism using digital watermarking in NoC-based SoCs. We consider a widely explored threat model in on-chip communication architectures where a hardware Trojan-infected router in the NoC IP copies packets passing through it, and re-routes the duplicated packets to an accompanying malicious application running on another IP in an attempt to leak information. Compared to existing authenticated encryption based methods, our approach offers significant performance improvement while providing the required security guarantees. Performance improvement is achieved by replacing authentication with packet watermarking that can detect duplicated packet streams at the network interface of the receiver. We discussed the accuracy and security of our approach using theoretical models and empirically validated them. Experimental results demonstrated that our approach can significantly outperform the state-of-the-art methods.

Acknowledgment

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

References

[1] A. Sodani *et al.*, "Knights landing: Second-generation intel xeon phi product," *IEEE MICRO*, vol. 36, no. 2, pp. 34–46, 2016.

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

[3] D. M. Ancajas *et al.*, "Fort-nocs: Mitigating the threat of a compromised noc," in *Proceedings of the 51st Annual Design Automation Conference (DAC)*. ACM, 2014, pp. 1–6.

[4] S. Charles *et al.*, "Real-time detection and localization of dos attacks in noc based socs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1160–1165.

[5] "The big hack: How china used a tiny chip to infiltrate u.s. companies," https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies.

[6] S. T. King *et al.*, "Designing and implementing malicious hardware." *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, vol. 8, pp. 1–8, 2008.

[7] M. Hussain *et al.*, "Eetd: An energy efficient design for runtime hardware trojan detection in untrusted network-on-chip," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 345–350.

[8] S. Charles *et al.*, "Lightweight anonymous routing in noc based socs," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 334–337.

[9] M. K. JYV *et al.*, "Run time mitigation of performance degradation hardware trojan attacks in network on chip," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 738–743.

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

[11] J. Sepúlveda *et al.*, "Towards protected mpsoc communication for information protection against a malicious noc," *Procedia computer science*, vol. 108, pp. 1103–1112, 2017.

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

[13] K. Sajeesh and H. K. Kapoor, "An authenticated encryption based security framework for noc architectures," in *2011 International Symposium on Electronic System Design*. IEEE, 2011, pp. 134–139.

[14] J. Porquet *et al.*, "Noc-mpu: A secure architecture for flexible co-hosting on shared memory mpsocs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2011, pp. 1–4.

[15] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 142–151.

[16] H. K. Kapoor *et al.*, "A security framework for noc using authenticated encryption and session keys," *Circuits, Systems, and Signal Processing*, vol. 32, no. 6, pp. 2605–2622, 2013.

[17] Q. Yu and J. Frey, "Exploiting error control approaches for hardware trojans on network-on-chip links," in *2013 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFTS)*. IEEE, 2013, pp. 266–271.

[18] A. Saeed *et al.*, "An id and address protection unit for noc based communication architectures," in *Proc. of the 7th International Conference on Security of Information and Networks*, 2014, pp. 288–294.

[19] J. Sepúlveda *et al.*, "Reconfigurable security architecture for disrupted protection zones in noc-based mpsocs," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2015, pp. 1–8.

[20] R. JS *et al.*, "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, 2015, p. 8.

[21] A. K. Biswas *et al.*, "Router attack toward noc-enabled mpsoc and monitoring countermeasures against such threat," *Circuits, Systems, and Signal Processing*, vol. 34, no. 10, pp. 3241–3290, 2015.

[22] C. Reinbrecht *et al.*, "Gossip noc–avoiding timing side-channel attacks through traffic management," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 601–606.

[23] N. Prasad *et al.*, "Runtime mitigation of illegal packet request attacks in networks-on-chip," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[24] J. Frey and Q. Yu, "A hardened network-on-chip design using runtime hardware trojan mitigation methods," *Integration*, vol. 56, pp. 15–31, 2017.

[25] L. S. Indrusiak *et al.*, "Side-channel attack resilience through route randomisation in secure real-time networks-on-chip," in *12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2017, pp. 1–8.

[26] J. Sepúlveda *et al.*, "Towards the formal verification of security properties of a network-on-chip router," in *23rd European Test Symposium (ETS)*. IEEE, 2018, pp. 1–6.

[27] S. V. R. Chittamuru *et al.*, "SOTERIA: Exploiting process variations to enhance hardware security with photonic NoC architectures," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[28] B. Lebiednik *et al.*, "Architecting a secure wireless network-on-chip," in *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1–8.

[29] L. S. Indrusiak *et al.*, "Side-channel protected mpsoc through secure real-time networks-on-chip," *Microprocessors and Microsystems*, vol. 68, pp. 34–46, 2019.

[30] A. K. Biswas, "Network-on-chip intellectual property protection using circular path–based fingerprinting," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 1, pp. 1–22, 2020.

[31] A. Iacovazzi *et al.*, "Network flow watermarking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 512–530, 2016.

[32] H. Deng *et al.*, "Selective forwarding attack detection using watermark in wsns," in *International Colloquium on Computing, Communication, Control, and Management (ISECS)*, vol. 3. IEEE, 2009, pp. 109–113.

[33] X. Wang *et al.*, "Robust network-based attack attribution through probabilistic watermarking of packet flows," North Carolina State University. Dept. of Computer Science, Tech. Rep., 2005.

[34] Z. Ling *et al.*, "Novel packet size-based covert channel attacks against anonymizer," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2411–2426, 2012.

[35] A. Houmansadr and N. Borisov, "Botmosaic: Collaborative network watermark for the detection of irc-based botnets," *Journal of Systems and Software*, vol. 86, no. 3, pp. 707–715, 2013.

[36] A. Houmansadr *et al.*, "Rainbow: A robust and invisible non-blind watermark for network flows." in *NDSS*, 2009.

[37] A. Zand *et al.*, "Rippler: Delay injection for service dependency detection," in *IEEE INFOCOM*, 2014, pp. 2157–2165.

[38] P. Mishra *et al.*, *Hardware IP security and trust*. Springer, ISBN 978-3-319-49024-3, 2017.

[39] Arteris, "Flexnoc resilience package," 2009, www.arteris.com/flexnoc-resilience-package-functional-safety.

[40] K. Shuler, "Majority of leading china semiconductor companies rely on arteris network-on-chip interconnect ip," 2013.

[41] F. Farahmandi *et al.*, *System-on-Chip Security Validation and Verification*. Springer, ISBN 978-3-030-30596-3, 2020.

[42] P. Mishra and S. Charles, *Network-on-Chip Security and Privacy*. Springer, ISBN 978-3-030-69130-1, 2021.

[43] S. C. Woo *et al.*, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.

[44] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[45] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[46] R. M. Roth and G. Seroussi, "Bounds for binary codes with narrow distance distributions," *IEEE transactions on information theory*, vol. 53, no. 8, pp. 2760–2768, 2007.

[47] I. Dumer *et al.*, "Hardness of approximating the minimum distance of a linear code," *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 22–37, 2003.

[48] M. Best *et al.*, "Bounds for binary codes of length less than 25," *IEEE Transactions on Information theory*, vol. 24, no. 1, pp. 81–93, 1978.

[49] N. Agarwal *et al.*, "Garnet: A detailed on-chip network model inside a full-system simulator," in *IEEE International symposium on performance analysis of systems and software*, 2009, pp. 33–42.

[50] A. Van Herrewege and I. Verbauwhede, "Software only, extremely compact, keccak-based secure prng on arm cortex-m," in *Proc. 51st Annual Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[51] M. Bakiri *et al.*, "Survey on hardware implementation of random number generators on fpga: Theory and experimental analyses," *Computer Science Review*, vol. 27, pp. 135–153, 2018.

[52] A. May and I. Ozerov, "On computing nearest neighbors with applications to decoding of binary linear codes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 203–228.

[53] S. Charles *et al.*, "Exploration of memory and cluster modes in directory-based many-core cmps," in *Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1–8.

[54] Intel, "Intel xeon phi processor 7210," https://ark.intel.com/content/www/us/en/ark/products/94033/intel-xeon-phi-processor-7210-16gb-1-30-ghz-64-core.html.

Subodha Charles is a Senior Lecturer in the Department of Electronics and Telecommunications Engineering, University of Moratuwa, Sri Lanka. He received his Ph.D in Computer Science from the University of Florida in 2020. His research interests include hardware security and trust, embedded systems and computer architecture.

Vincent Bindschaedler is an assistant professor in the department of Computer and Information Science and Engineering at the University of Florida. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2018. His research interests include data privacy, applied cryptography, and privacy-preserving technologies. His recent work focuses on emerging problems at the intersection of machine learning with security and privacy.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. He received his Ph.D. in Computer Science from the University of California at Irvine in 2004. His research interests include embedded systems, hardware security and trust, system-on-chip validation, and quantum computing. He currently serves as an Associate Editor of ACM Transactions on Embedded Computing Systems and IEEE Transactions on VLSI Systems. He is an IEEE Fellow and ACM Distinguished Scientist.