2019

Design of secure and trustworthy system-on-chip architectures using hardware-based root-of-trust techniques

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

DESIGN OF SECURE AND TRUSTWORTHY SYSTEM-ON-CHIP ARCHITECTURES USING HARDWARE-BASED ROOT-OF-TRUST TECHNIQUES

by

LAKE BU

B.S., Wuhan University, 2006
M.S., Boston University, 2008

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2019

Approved by

First Reader

Michel A. Kinsy, PhD
Assistant Professor of Electrical and Computer Engineering

Second Reader

Mark G. Karpovsky, PhD
Professor of Electrical and Computer Engineering

Third Reader

Ari Trachtenberg, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Fourth Reader

Qiaoyan Yu, PhD
Assistant Professor of Electrical and Computer Engineering
University of New Hampshire

The LORD is my shepherd; I shall not want.
He makes me lie down in green pastures. He leads me beside still waters. He restores my soul. He leads me in paths of righteousness for his name's sake.

– Psalm 23:1-3

Acknowledgments

First, I would like to sincerely thank my PhD advisor Michel A. Kinsy. Professor Kinsy sets up a role model for me in academic area with his tireless and hardworking attitude. He always encourages me to work for excellence in every detail. He also sets up a good example in faithfully doing everything that needs to be done, no matter if he likes it or not. Professor Kinsy cares a lot about me and my family in many aspects, such as giving me advices in family life and inviting my family to his home. Although he is strict in research as a professor, he is also always ready to help as a great mentor. I can never forget the evening which was his birthday, but professor Kinsy stayed late with me to help me with my defense preparation. Sincerely, I have been so blessed to be his student and mentored by him. If I can choose again, I would still choose to be his student.

Second, I want to sincerely thank professor Mark G. Karpovsky, who supervised me from 2014-2016 as my academic advisor. Professor Karpovsky guided me into the research field and taught me abundant knowledge on coding theory, which later became the most important mathematical tool in my PhD research. He sets up a great example for me by working with me together to solve each problem on scratch papers and white boards, even on his birthday. Professor Karpovsky has great insights in his field and always provided me with valuable advices in both the research perspectives and detailed problem solving. He is also very caring to my personal career and always supports me unconditionally.

Then, I would like to mention that my lab mates, Rashmi Agrawal, Mihailo Isakov, Novak Boskov, Alan Ehret, Miguel Mark, Shanshan Wang, and Sahan Bandara, who offered me great help during my thesis writing and defense rehearsal by spending a great amount of time in listening to my practice and proofreading my thesis draft.

Finally, I want to dedicate this thesis to my dear family. They have provided

precious support during my thesis writing and defense preparation. My parents (Jun Fan and Jinfeng Lei) took care of house chores and gave wholehearted love to my wife Rachel, my son Eli, and me. Without them I will not be able to finish my last year of PhD.

My wife Rachel always supported me with her full understanding and encouragement when I had to go home late or even stay up for the whole night in the lab. She is such a great wife that always offers her unconditional and sacrificial love to me. It is my greatest blessing to spend the rest of my life with her.

Lake Bu

PhD student

ECE Department of Boston University

DESIGN OF SECURE AND TRUSTWORTHY SYSTEM-ON-CHIP ARCHITECTURES USING HARDWARE-BASED ROOT-OF-TRUST TECHNIQUES

LAKE BU

Boston University, College of Engineering, 2019

Major Professor: Michel A. Kinsy, PhD
Assistant Professor of Electrical and Computer Engineering

ABSTRACT

Cyber-security is now a critical concern in a wide range of embedded computing modules, communications systems, and connected devices. These devices are used in medical electronics, automotive systems, power grid systems, robotics, and avionics. The general consensus today is that conventional approaches and software-only schemes are not sufficient to provide desired security protections and trustworthiness.

Comprehensive hardware-software security solutions so far have remained elusive. One major challenge is that in current system-on-chip (SoCs) designs, processing elements (PEs) and executable codes with varying levels of trust, are all integrated on the same computing platform to share resources. This interdependency of modules creates a fertile attack ground and represents the Achilles' heel of heterogeneous SoC architectures.

The salient research question addressed in this dissertation is "can one design a *secure* computer system out of *non-secure* or *untrusted* computing IP components and cores?". In response to this question, we establish a generalized, user/designer-centric

set of design principles which intend to advance the construction of secure heterogeneous multicore computing systems. We develop algorithms, models of computation, and hardware security primitives to integrate secure and non-secure processing elements into the same chip design while aiming for: (a) maintaining individual core's security; (b) preventing data leakage and corruption; (c) promoting data and resource sharing among the cores; and (d) tolerating malicious behaviors from untrusted processing elements and software applications.

The key contributions of this thesis are:

1. The introduction of a new architectural model for integrating processing elements with different security and trust levels, i.e., secure and non-secure cores with trusted and untrusted provenances;
2. A generalized process isolation design methodology for the new architecture model that covers both the software and hardware layers to (i) create hardware-assisted virtual logical zones, and (ii) perform both static and runtime security, privilege level and trust authentication checks;
3. A set of secure protocols and hardware root-of-trust (RoT) primitives to support the process isolation design and to provide the following functionalities: (i) hardware immutable identities – using physical unclonable functions, (ii) core hijacking and impersonation resistance – through a blind signature scheme, (iii) threshold-based data access control – with a robust and adaptive secure secret sharing algorithm, (iv) privacy-preserving authorization verification – by proposing a group anonymous authentication algorithm, and (v) denial of resource or denial of service attack avoidance – by developing an interconnect network routing algorithm and a memory access mechanism according to user-defined security policies.
4. An evaluation of the security of the proposed hardware primitives in the post-

quantum era, and possible extensions and algorithmic modifications for their post-quantum resistance.

In this dissertation, we advance the practicality of secure-by-construction methodologies in SoC architecture design. The methodology allows for the use of unsecured or untrusted processing elements in the construction of these secure architectures and tries to extend their effectiveness into the post-quantum computing era.

Contents

List of Tables

List of Figures

List of Abbreviations

AC	.....	Access Code
BEL	.....	Basic Element
CRP	.....	Challenge-Response Pairs
GTB	.....	Group Testing Based codes
IC	.....	Integrated Circuit
IP	.....	Intellectual Property
KEX	.....	Key Exchange
LUT	.....	Look Up Table
MDS	.....	Maximum Distance Separable
Mi-PUF	.....	Multi-Identity Physical Unclonable Functions
MMU	.....	Memory Management Unit
NI	.....	Network Interface
NIST	.....	National Institute of Standards and Technology
NOC	.....	Network-on-Chip
NTT	.....	Number-Theoretic Transform
NVM	.....	Non-Volatile Memories
NWC	.....	Negative Wrapped Convolution
OLSC	.....	Orthogonal Latin Square Codes
OT	.....	Oblivious Transfer
PE	.....	Processing Element
PKC	.....	Public-key Cryptosystem
PUF	.....	Physical Unclonable Functions
R-LWE	.....	Ring-Learning with Errors
RO	.....	Ring Oscillator
RoT	.....	Root of Trust
RS	.....	Reed-Solomon
SoC	.....	System on Chip
SCC	.....	Self-Checking Checker
TRNG	.....	True Random Number Generator
ZKP	.....	Zero-Knowledge Proof

Chapter 1

Introduction

Heterogeneous system-on-chip (SoC) architectures have many advantages. They generally consist of highly specialized application-specific processing units and general-purpose cores. Their performance and power can be better optimized, and the specialization of compute units to different tasks promises greater energy/area efficiency. For example, (Kumar et al., 2004) show that a heterogeneous computer architecture can outperform a comparable-area homogeneous architecture by up to 63%.

By integrating cross-vendor chipsets and software onto one platform, heterogeneous system-on-chip (SoC) architectures allow users or programmers to take advantage of a whole suite of distinct functionalities provided by different manufacturers and intellectual property (IP) providers.

An equally important fact is that, the design of these systems and the development of associated kernels and applications are increasingly global. As shown by the Semiconductor Industry Association (SIA) report in Fig. 1·1, at present the top participants of the semiconductor industry (manufacturing, fabrication, and packaging companies, etc.) come from more than 23 countries on 3 continents (Asia, North America, and Europe). A single semiconductor production process can involve at least 4 countries and 3 trips around the world. As a result, the provenances of IPs become harder to establish.

In addition, the runtime interactions on those heterogeneous SoC systems are fairly complex. Processes may share processing elements, data, and I/O processing

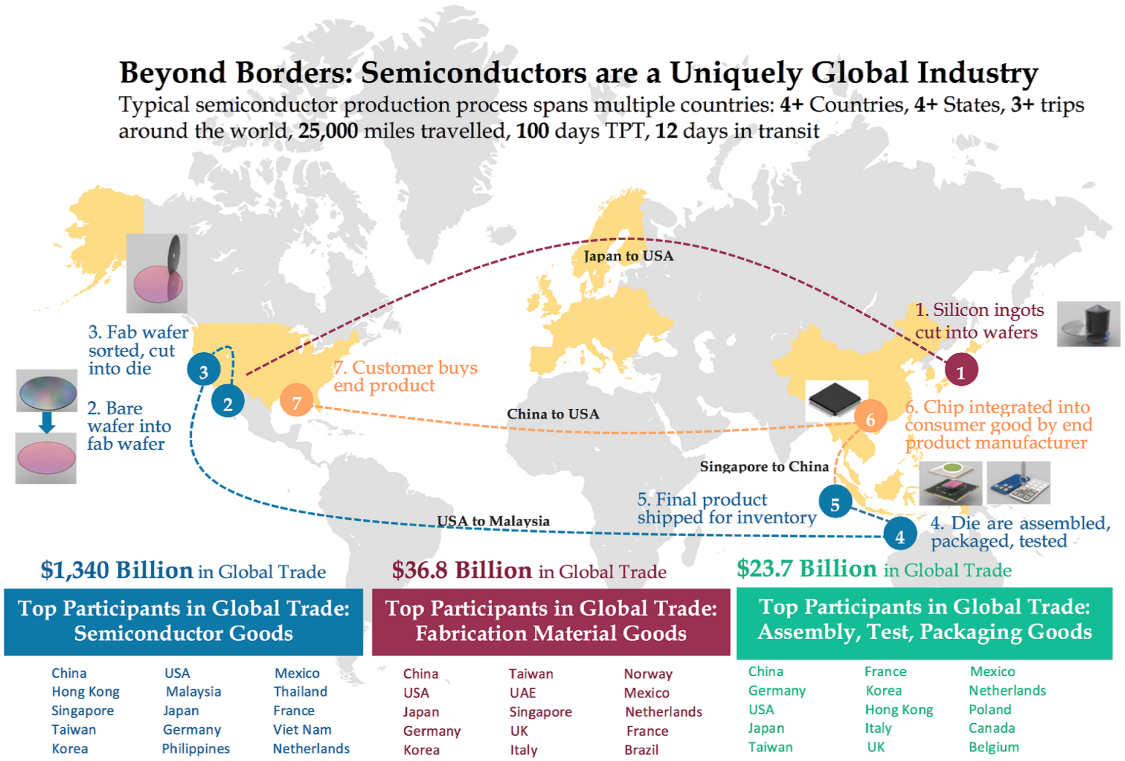


Figure 1·1: The global supply chain map from the Semiconductor Industry Association ((SIA), 2018).

time. Similarly, processing elements/processors/cores themselves may share cache or memory structures, network-on-chip resources, and I/O modules.

In all, despite the many advantages of heterogeneous SoC systems, e.g., higher performance, better energy efficiency, lower communication latencies, they also give rise to a number of critical security challenges. One, the globalization of the semiconductor industry has made IP provenance checking more challenging. Second, runtime interactions and resource sharing of processing elements have created a fertile attack ground and represents the Achilles' heel of heterogeneous SoC architectures.

1.1 Security Challenges in Current SoCs and Distributed Systems

We identify three security challenges in current heterogeneous SoCs, which motivate the work in this dissertation.

1.1.1 Trust Issues in the Supply Chain

As mentioned in the beginning of this section, most heterogeneous SoCs may consist of processing elements from different IP providers or manufacturers. Also the applications running on them may have varying levels of security and trust, all executing on the same compute platform while sharing resources with each other.

For example, in a multicore smartphone chip, complex runtime interactions between processors running untrusted applications can sometimes circumvent the built-in security guards, in order to access memory blocks in protected sections of the phone. These systems are vulnerable to a variety of software-centric attacks, such as spyware, trojans and viruses, as well as hardware-centric attacks such as side channel attacks and hardware trojans (Tehranipoor and Koushanfar, 2010).

There have been several research efforts exposing the security issues in the global supply chain. As (Salmani, 2018) pointed out, in semiconductor industry, the time to market window is made as short as possible in order to maximize the revenue. Therefore, design strategies based on intellectual property (IP) reuse and manufacture outsourcing are widely adopted. For this very reason, vulnerabilities in the IPs and trojans inserted in the manufacturing process (hardware design, fabrication, packaging etc.) have turned into a growing concern. Worse, since most semiconductor companies are focusing on design only and becoming "fab-less", malicious hardware modifications and trojans can be inserted at fabrication time by untrusted foundries (Forte et al., 2016). For example, according to the recent reports (International De-

fense Security and Technology (IDST), 2019) (Zorz, 2018), counterfeits and trojans are the two major causes of hardware vulnerabilities in the U.S. military systems.

On the other hand, realizing that most semiconductor designers and users have to live with an untrusted supply chain, there is a trend to move away from relying only on "trusted foundries", to "security by design and chain of custody" (Lapedus, 2018). The U.S. Department of Defense (DoD) released their view on approaches to ensure trusted semiconductor products, as shown in Fig. 1·2. This flow diagram covers a wide spectrum from design to operation, assuming injection of malicious conducts can happen during any of the production stages.

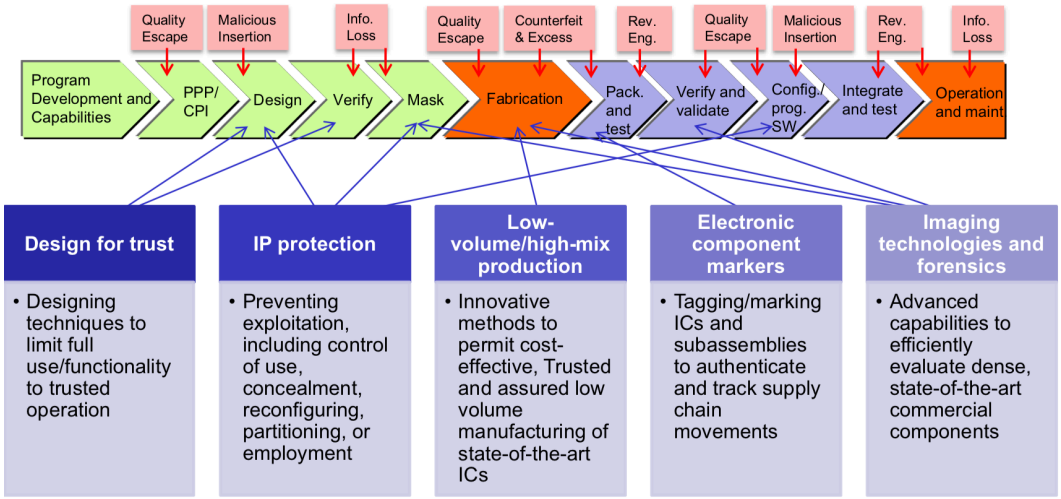


Figure 1·2: New Trust and Assurance Approaches. Source: DoD (Lapedus, 2018).

1.1.2 Insufficient Security by Software-only Protections

A general consensus today is that conventional approaches and software-only add-on schemes have failed to provide sufficient security protections and trustworthiness. As Defense Advanced Research Projects Agency (DARPA) showed in their brief (Fig. 1·3), it is increasingly expensive to defend by software-only solutions than to attack

(Kaufman, 2011).

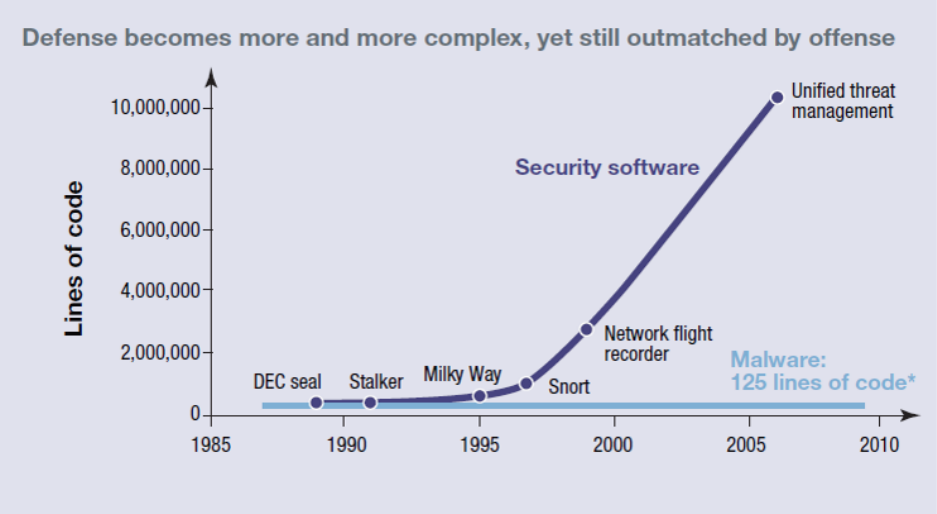


Figure 1·3: Defense Advanced Research Projects Agency (DARPA): Brief to Defense Science Board (DSB) Task Force (May 2011).

In addition, some hardware security issues are beyond the capability of software-based protections. In an analysis by IoT Inspector on over 4,000 embedded devices from 70 different hardware vendors, they found 580 cryptographic secret keys shared between a great number of devices (Khandelwal, 2015). Not only have some "lazy" manufacturers used identical keys among their own devices, but also there are keys shared amongst different vendors.

Another alarming fact is that, when the researchers scanned the Internet for those 580 keys, they found that at least 230 of them are actively used by more than 4 Million IoT devices all over the world, with the United States being the top affected country (by owning 26.3% of those devices).

These carelessly shared secret keys can be used to forge counterfeits which can deceive authentication and breach secure systems. A similar report (Byrne, 2015) stressed that, since all these keys are hard-coded into the IoT devices, this issue cannot be resolved using software patches.

In a report titled *Is Computer Security Becoming a Hardware Problem?* (Byrne, 2016), Kocher, the co-author of the SSL v3.0 protocol, stated that: "To make progress, we need another building block: simple, high-assurance hardware for secure computation."

As a matter of fact, there have been increasing proposals to utilize hardware as the root of trust (RoT) for critical tasks such as authentication, key storage, and key transmission etc. The Computer Security Resource Center (CSRC) in National Institute of Standards and Technology (NIST) launched a project to study the use of hardware RoT in securing computing systems (NIST, 2018). Companies such as Synopsis and Intel have also been exploring this field (Elias, 2017) (Intel, 2017) even before NIST. The advantage of this approach is that it can provide functions that malware cannot tamper with. A system equipped with hardware RoT typically cannot be breached unless the attackers gain physical access to it.

1.1.3 Introduction of General-Purpose Quantum Computers

In the last three years, we have witnessed a number of breakthroughs and several key milestones towards the development of general-purpose quantum computers. These advances do bring with them critical challenges to classical cryptosystems, due to the existing quantum algorithms to solve certain security reduction problems on which the classical cryptosystems rely.

Shor's algorithm (Shor, 1999) leveraging quantum Fourier transform is able to solve the integer factorization problem efficiently. Therefore, the current popular cryptographic algorithms such as RSA, ElGamal, Diffie-Hellman, and elliptic-curve cryptography, which rely on the hardness of integer factorization and discrete logarithms, are now subject to attacks by quantum computers. Moreover, for symmetric cryptography, Grover's algorithm (Grover, 1996) applies fast search in the key space, so that the security level of symmetric encryptions can be reduced to the square root

of its original (e.g., the 128-bit and 192-bit AES now only have 64-bit and 96-bit security levels, failing to meet the 112-bit minimum security level recommended by NIST).

Since the possibility of using quantum effects in computation was brought up by (Feynman, 1959), numerous efforts have been dedicated to realize and even commercialize the quantum computers, especially in the past three years.

From late 2017 to early 2018, technology companies such as IBM, Intel, and Google, announced their construction and testing of 50-, 49-, and 72-qubit computers respectively (Knight, 2017) (Hsu, 2018) (Courtland, 2017). In July 2018, for the first time in the world, the University of Sydney accomplished a multi-qubit computation on a system of trapped ions, which is believed to be one of the leading platforms in building universal quantum computers (Science-Daily, 2018). In December 2018, IonQ claimed to have built a quantum computer with 160 qubits, which was referred to as the "best quantum computer yet" by reporters (Mandelbaum, 2018). In January 2019, IBM released its first 20-qubit commercial quantum computer: Q System One (Aron, 2019). Besides the boost in physical implementation of quantum computers, breakthroughs in the verification of quantum computation were achieved in 2017 (Aharonov et al., 2017), and more efficient error correction schemes have been proposed recently as well (Michael et al., 2016).

NTRU Innovation, a cryptographic company dedicated to developing quantum-resistant cryptosystems, made a projection of the arrival of universal quantum computers in late 2015 (Fig. 1·4). Based on the milestones achieved then, NTRU predicted that within 5 to 10 years, powerful quantum computers may become accessible (Wilson, 2016). This also matches the time line given by other institutes such as Microsoft Quantum and NIST.

Another interesting attempt to apply Moore's law to the growth of qubits, also

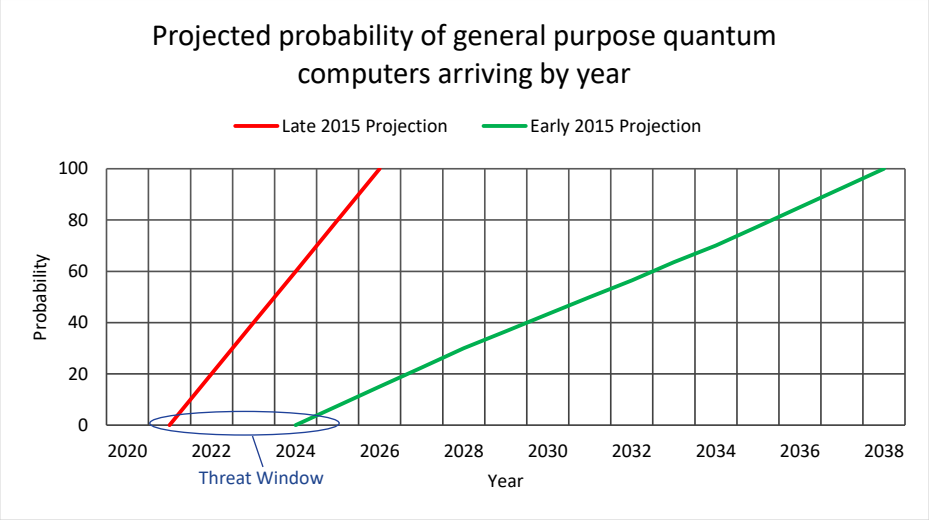


Figure 1·4: NTRU says critical infrastructure "MUST be re-tooled when the threat window opens" (Wilson, 2016). As more breakthroughs are achieved in the past three years in the physical implementation of quantum computers, the threat window may move even closer to us.

shows a similarly estimated time line in breaking the RSA scheme within 5 to 10 years (Quantum Computing Report, 2017), as shown in Fig. 1·5.

These estimations indicate that, it is feasible that within the next decade, secure systems based on classical cryptographic algorithms, e.g., most banking, government, or medical systems, will no longer be considered trustworthy.

In response to the looming threat of the quantum computers, a number of new cryptosystems have been proposed for the post-quantum era. In early 2017, NIST launched a standardization campaign for post-quantum cryptography and 69 candidates were submitted. On January 30, 2019, 27 candidates were selected for the second round (semi-final) of this contest (NIST, 2019). Among all these submissions, lattice-based (12 candidates) and code-based (8 candidates) cryptosystems are leading candidates. However, efficient and secure hardware implementation of these algorithms still remains a young research field.

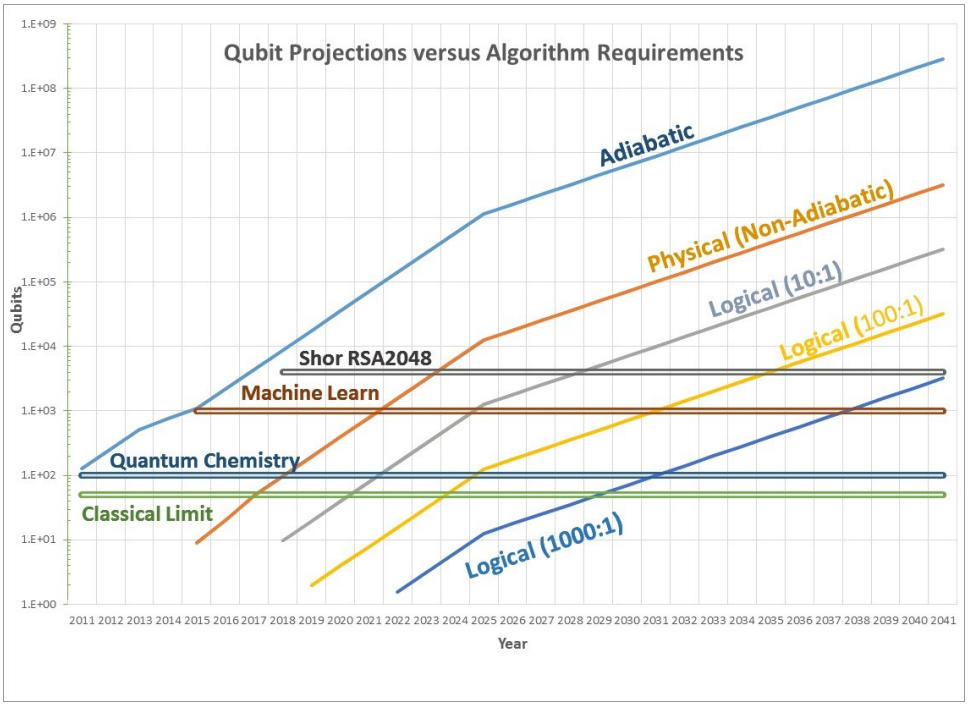


Figure 1·5: Projected qubit growth with different quantum computer mechanisms (Quantum Computing Report, 2017).

1.2 Thesis Contributions

The question we aim to address in this dissertation is "can one design a *secure* computer system out of *non-secure* or *untrusted* computing IP components and cores?". Considering the previous discussions on security challenges in SoCs, this question can break down to: (1) how to protect heterogeneous processing elements (PEs) with different security levels; (2) how to optimally and securely share resources and data among those processing elements; (3) how to anchor the system security on a strong root-of-trust and construct secure protocols to resist PE's malicious behaviors; and (4), how to extend the usability of the system to the post-quantum era.

Therefore, to address these design challenges, in this thesis we propose a new architectural model. The major technical contributions of this approach are:

1. The introduction of **a new architectural model** for integrating processing

elements with different security and trust levels, i.e., secure and non-secure cores with trusted and untrusted provenances;

2. A generalized **process isolation** design methodology (Kinsy et al., 2018) for the new architecture model that covers both the software and hardware layers to (i) create hardware-assisted virtual logical zones, and (ii), perform both static and runtime trust level verification.

   Specifically, we group the processing elements (PE) into virtual logical zones called *islands* based on each PE's dynamic trust level. Process and data isolation are thus established based on the partition by *islands*. Data/resource sharing is allowed within an *island* but prohibited across *islands*. A set of *island join* and *leave* protocols are enforced to verify the legitimacy of each memory access request.

3. **Hardware-based root-of-trust:** we anchor the security of the SoC system on hardware-based root-of-trust (RoT), which consists of:

   - Hardware-assisted process isolation: a hardware Network Interface (NI) module is constructed to facilitate the dynamic *island* forming by 1) handling the memory access requests; 2) verifying the privilege level of each request; and 3) generating *island* session keys upon each *island* membership change;
   - Hardware immutable identities: we use Physical Unclonable Functions (PUFs) to perform PE authentication, identification, and key storage/-transmission (Bu et al., 2018a) (Bu et al., 2018b). We also propose a Multi-identity PUF (Mi-PUF) design (Bu and Kinsy, 2018a) which enables an individual PE to be labeled with multiple identities upon its different activities (memory accesses or *island* memberships).

4. A set of **secure protocols** based on hardware root-of-trust to strengthen the process isolation. They provide the following functionalities:

    - Core impersonation resistance: using a blind signature scheme to uniquely bound each core's memory access permission to itself, so that the memory access activity cannot be spoofed (Kinsy et al., 2018);
    - Threshold authorization to data resources: using a robust adaptive secure secret sharing algorithm to prevent dishonest "gatekeepers" of an *island* from unfaithfully authorizing *island join* to disqualified PEs (Bu et al., 2018c) (Bu et al., 2017);
    - Privacy-preserving privilege verification: through a group anonymous authentication algorithm to verify each PE's trust level without revealing its identity;
    - Denial of service attack avoidance: by developing a security-aware interconnect network routing algorithm and a memory access mechanism based on user-defined security policies (Kinsy et al., 2018).

5. To construct the secure protocols in 2 and 4, we propose to use a set of **quantum-resistant primitives** as the basic building blocks: public-key cryptosystem (PKC), key exchange (KEX), oblivious transfer (OT), and zero-knowledge proof (ZKP) (Bu et al., 2019). Their features are:

    - All primitives are based on the Ring-Learning with Errors (R-LWE) cryptosystems. R-LWE as a variation of lattice-based cryptosystems, is one of the most promising candidates of NIST's post-quantum cryptography standardization;
    - Novel algorithms for simple oblivious transfer and zero-knowledge proof based on post-quantum cryptography (Bu et al., 2019);

- A fully parameterizable hardware design of these primitives;
- All primitives can be reused to compose more complex secure systems for the post-quantum era.

1.3 Thesis Outline and Chapter Summaries

The dissertation aims to construct from untrusted elements trustworthy heterogeneous SoCs with certain security features. We establish a set of design principles to enable process isolation, and a set of hardware RoT-based protocols to defend against malicious processing elements (PEs). We now outline each of the chapters.

Chapter 2: Generalized Framework for Designing Secure SoC Architectures from Untrusted Components

We begin by introducing a new architectural model which consists of a set of design principles to build trusted heterogeneous systems from untrusted components.

As mentioned previously in the thesis contribution, this new architectural model addresses virtual channel and memory attacks through process isolation, by grouping processors into virtual logical zones called *islands*. It also inherits and extends the well-established concept of session keys, which enable elements within an insecure network to establish secure communications (Katz and Shin, 2005).

First, the on-chip PEs are divided into *wards* (physical zones for PE residence) that are identified and formed at system integration time, based on IP or PE provenance. *Wards* are created to help negotiate the security keys used to create the *islands* in a trusted manner. Because the security level is inherent to the IP origin, the *wards* remain constant throughout the chip's lifetime.

Second, the PEs in the new architectural model are virtually grouped into logical zones called *islands* based on their static or runtime security characteristics. These nodes can be either secure processors or non-secure processors, as well as any combi-

nation of hardware processing units with varying levels of trust. Figure 1·6 shows an illustration of a virtually partitioned *islands* on a multicore chip. Each PE is assigned to an island based on both trust levels of the PE and the applications running on the PE. PEs within the same *island* will be able to share data, but resource sharing across *islands* is prohibited.

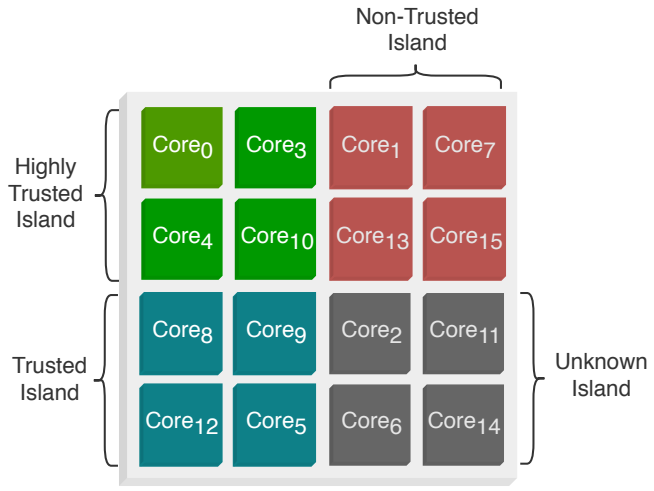


Figure 1·6: Four virtual *islands* partitioning based on PEs' dynamic security characteristics: highly trusted, trusted, untrusted and unknown.

One of the key innovations of the proposed architectural model is that the physical *wards* and the virtual *islands* are decoupled, hence making the node placement decisions independent of the processing cores' security needs. This allows for efficient on-chip routing and node grouping.

The architectural model coordinates the virtual *island* partitioning through a secure hardware module — the network interface (NI). The NI serves as the enforcer of the user-defined security policy by facilitating the dynamic *island* forming. The NI verifies if a memory access request is valid according to its access code (AC), and authorizes *island join* if the request is found to be legitimate.

In addition, we propose a set of secure protocols to:

(1) tailor the AC for each request using a blind signature scheme, so that the same AC cannot be reused by other untrusted elements;

(2) authorize the *island join* in a group decision manner by a cheater-tolerant secret sharing scheme, so that a dishonest sponsor nodes cannot jeopardize the *join* authorization;

(3) enable an incognito *join* mode with oblivious transfer algorithms, in order to preserve PEs' privacy on data requests;

(4) perform privilege verification without revealing a PE's identity by a group anonymous authentication protocol.

These secure protocols are constructed based on a number of well-established cryptographic primitives such as physical unclonable functions (PUF), true random number generator (TRNG), public-key cryptosystem (PKC) and oblivious transfer (OT). They will be introduced in detail in Chapters 3, 4, and 5.

Chapter 3: Hardware Root-of-Trust Security Primitives

In this chapter we introduce the design and implementation of the physical unclonable functions (PUF), which will serve as a hardware root-of-trust primitive for many of the secure protocols in this thesis. We use PUF as the unique identifier, authentication and key storage primitive of the PEs in SoC. PUF works by its challenge-response pairs (CRP). Before a hardware element with a PUF is released, the verifier will feed the PUF with multiple inputs called "challenges". The outputs are collected and stored as "responses". A challenge and its corresponding response form a CRP. Upon authentication, the verifier sends a challenge to the element's PUF. By comparing the returned response with the stored CRP, the verifier is able to determine the authenticity of the element.

PUF is built by amplifying the unique manufacturing variations of a piece of hardware. Such variations can be the slight differences on transistor sizes, drain currents, threshold voltages, or capacitance. Since PUFs take advantage of the natural characteristics of circuits, they are also called "hardware fingerprints".

There are a couple of advantages to use such a hardware primitive as the RoT instead of using a static string (a secret key or an ID tag). First, PUFs do not require expensive non-volatile memories (NVM) for secret key storage. Second, PUFs are naturally integrated in circuits and cannot be separated from them to make counterfeits. Third, as CRPs are independent from each other, capturing one does not impact the security of others.

In this chapter, we first explore different types of PUFs and choose the delay-based PUF to construct the RoT. We then dive into the implementation details of PUFs on the FPGA platform. We provide a discussion on the common mistakes made by PUF designers, and related mitigations. In the end, we propose a novel type of PUF named Multi-identity PUF (Mi-PUF). Unlike ordinary PUFs which attest only one legal identity of an element, the Mi-PUF is able to attach multiple legitimate identities to a single hardware element. Mi-PUFs can be used to assign multiple identities to the same PE to label its different activities (memory accesses or *island* memberships).

Chapter 4: Towards Programmable All-Digital True Random Number Generators (TRNG) for Cryptographic Functions

TRNG are crucial for cryptosystems. They determine the quality of secret keys and random vectors. In the proposed architectural model, true random numbers are used in a number of modules such as the public-key system and the majority of the secure protocols. To serve these modules for different purposes (high entropy, high throughput, low power etc.), we design a programmable TRNG using ring oscillators

and Lorenz chaotic maps.

Chapter 5: Quantum-resistant Extension of Hardware Primitives

In order to construct the secure protocols in the proposed architectural model, we need two hardware primitives: the public-key cryptosystem (PKC) and oblivious transfer (OT). While there are abundant hardware implementation resources for classical cryptosystems such as RSA or ECC, we choose to implement the quantum-resistant cryptosystems for the reasons mentioned in Section 1.1.3: the proposed new architectural model can remain trustworthy if universal quantum computers ever become feasible.

The semi-final of NIST's post-quantum cryptography standardization contest officially started in March 2019 (NIST, 2019). Out of 27 candidates, 12 are Ring-Learning with Error (R-LWE)-based, 8 code-based, 4 multivariate-based (all under digital signature category), and 3 others. It is clear that the R-LWE and code-based (McEliece) cryptosystems have a high chance of winning the contest. Therefore, in this thesis we implement the hardware primitives based on the R-LWE cryptosystem, and will explore the code-based cryptosystem primitives in the future work.

Although there are a myriad of works exploring different implementations of the Ring-LWE algorithm in software, hardware level design space exploration efforts have been very timid. Moreover, out of the existing hardware implementations, very few of them focus on scalability and efficiency. One technical reason is that large finite field operations (thousands of bits for one vector) – which form the core computational kernel of both algorithms – remain a key challenge for many hardware designers.

To address this design challenge, we introduce a set of highly-optimized, parameterizable quantum-resistant hardware primitives for design space exploration of post-quantum cryptosystems. These hardware primitives include four frequently used security components: the public key cryptosystem (PKC), key exchange (KEX), oblivious

transfer (OT), and zero-knowledge proof (ZKP).

The major contributions of this hardware primitive toolbox are:

- *Algorithm:* Novel proposals of the quantum-resistant OT and ZKP primitives;
- *Parameterization:* A parameterizable design to generate variable-sized primitives to enable their deployment in small devices such as IoT devices, as well as large computing platforms such as homomorphic encryption engines;
- *Implementation:* FPGA-tailored optimization to reduce the area and power cost.

These primitives can serve as the fundamental building blocks to aid hardware designers in constructing larger secure systems for the post-quantum era.

Chapter 6: Experiments and Evaluation

For the system performance and evaluation, we implement an 8×8 2-D mesh topology design on a Xilinx FPGA device. For the power estimates, we use the Xilinx Power Estimator (XPE) in the Vivado Design Suites. The power numbers are the post-routing estimates using a vector based switching activity format (i.e., SAIF). The process feature was set to maximum and the power supply to default.

The *Heracles* (Kinsy et al., 2013) and the *BRISC-V* (Bandara et al., 2019) RTL simulators are used for all the experiments. Heracles' injector cores are used to create network and memory traffic. The trust level per core is assigned arbitrarily. The synthetic benchmarks and the SPLASH-2 benchmark suite are used to evaluate the new architectural model's performance and security enforcement.

We show through experimental results that, while performing the correct functionality under protection, the new architectural model adds hardware (area) overhead of no more than 17%, and no significant performance (throughput and latency) penalty.

Chapter 7: Conclusion and Feature Works

In this chapter, we conclude the dissertation and propose some of the future work on heterogeneous SoC systems.

Chapter 2

Generalized Framework for Designing Secure SoC Architectures from Untrusted Components

2.1 Introduction

In this chapter we discuss the core of the proposed new architectural model, which provides a set of design principles to construct secure heterogeneous systems from untrusted components (Kinsy et al., 2018).

The current trend in system-on-chip (SoCs) designs is system-level integration of heterogeneous technologies on the same chip. The design of these systems and the development of associated kernels and applications are increasingly global (Oberg et al., 2013). System designers and users of integrated circuits (ICs), intellectual property (IP), and SoCs are increasingly facing trust issues. In these designs, the processing elements may come from different providers, and application executable code may have varying levels of security and trust, all executing on the same compute platform and sharing resources. This creates a fertile attack ground and represents the Achilles' heel of heterogeneous SoC architectures and distributed connected devices. The general consensus today is that conventional approaches and software-only add-on schemes have failed to provide sufficient security protections and trustworthiness.

2.1.1 Security Problems

Heterogeneous system-on-chip designs are vulnerable to a variety of software-centric and hardware-centric attacks, such as spyware or viruses at the software level and side channel analysis or trojans at the hardware level (Tehranipoor and Koushanfar, 2010). On SoC architectures consisting of multiple cores, the runtime interactions between processing elements can be very complex and difficult to fully analyze at design time. As a result, processors running untrusted applications can sometimes circumvent the built-in security guards and access memory blocks in encrypted sections of the system (Chen et al., 2014).

These SoC architectures generally have: (a) a set of heterogeneous processing elements; (b) a memory subsystem; and (c) an interconnect network. For scalability reasons, network-on-chip (NoC) is broadly used as the communication fabric in these systems (Jerger and Peh, 2009). Unfortunately, the NoC is not immune to attacks. In fact, it is one of most targeted parts of the architecture, because it serves as the gateway to the other modules in systems (i.e., processing elements and memory units).

An adequate solution towards secure SoCs needs to provide: (i) a provable mechanism for robust isolation of hardware subsystems and program code (e.g., trusted vs. untrusted); (ii) efficient and fast access control to system resources (e.g., physical memory and routing paths); and (iii) support for user-defined security policies.

2.1.2 Threat Models

Heterogeneous multicore systems can be vulnerable to several attacks (Fiorin et al., 2007), including invasive attacks against hardware modules (using micro probing or other similar techniques), non-invasive attacks such as side channel attacks, Trojans and malware. In this work, we assume the attacker (malicious processes or untrusted PEs) has the following capabilities:

(a) The attacker is able to inject a deluge of useless packets into the network, in order to attempt on-chip denial of service (OC-DoS) attacks;

(b) The attacker is able to plow shared virtual channels (VC) and build their packet contents out of other flows' residual data;

(c) The attacker is able to leverage direct memory accesses (DMA) to attempt to read/write the physical memory;

(d) The attacker is able to apply cache side-channel attacks;

(e) The attacker is able to leverage transient executions (speculative or out-of-order executions) to attempt to bypass privilege checks.

We also assume the limitations of attackers:

- The attacker does not have physical access to the chip once it is integrated;
- The attacker cannot modify the kernel or operating system (OS);
- The attacker cannot access the system programming port;
- The attacker cannot manipulate or apply side-channel attacks to the memory management unit (MMU);
- The attacker cannot perform online re-multiplexing or manipulate the programmable logic controller (PLC) to re-multiplex the I/O pins.

The design methodology behind the new architectural model's design principles is to provide hardware-assisted mechanisms for user-defined security rules and their enforcement in heterogeneous many-core SoCs. It effectively decouples the trust level management of processing cores from the integrated SoC by process isolation. This

isolation is achieved through 1) virtualized partitioning based on cores' runtime trust levels, and 2) enforcing privilege checks on memory accesses.

Regarding the attack model, this new architectural model is geared towards providing the following key protections:

(A) A security-aware on-chip routing to cooperate with the virtualized process isolation, so that maliciously injected junk packets by untrusted processing elements (PEs) will not jam the traffic of trusted PEs, which are virtually partitioned into other virtual zones;

(B) The security-aware on-chip routing ensures that different paths are assigned to PEs of different trust levels. Also, sensitive communications between cores are encrypted. Thus, an untrusted PE cannot plow any confidential information from a trusted PE's virtual channel;

(C) With the trust level verification enforced on memory access requests, including the ones from physical I/O ports, no PE or external device is able to bypass this check to issue direct memory access (DMA);

(D) The virtual isolation allocates different memory regions for processes in different trust levels. Therefore, the isolation works as a hardware-assisted memory coloring, which effectively prevents untrusted processes from sharing the same cache sets with trusted processes. This approach guards the memory sub-system from certain classes of side-channel attacks;

(E) In addition, due to the trust level verification enforced on memory access requests, attacks leveraging speculative execution (Meltdown- or Spectre-like) will not be able to bypass privilege checks.

It is notable that, the proposed architectural model places no restrictions on the provenance or trust level of PEs. Thus, the model does not address the vulnerabilities

inherited from the PEs, such as side-channel attacks to page table pages leveraging the vulnerability of MMU, or Foreshadow-like attacks leveraging the L1 terminal fault of certain type of processors.

2.1.3 Overview of the Chapter

A high-level view of the proposed architectural model is shown in Fig. 2·1. It is an integration template allowing designers to include processing elements of various trust levels while still maintaining their security.

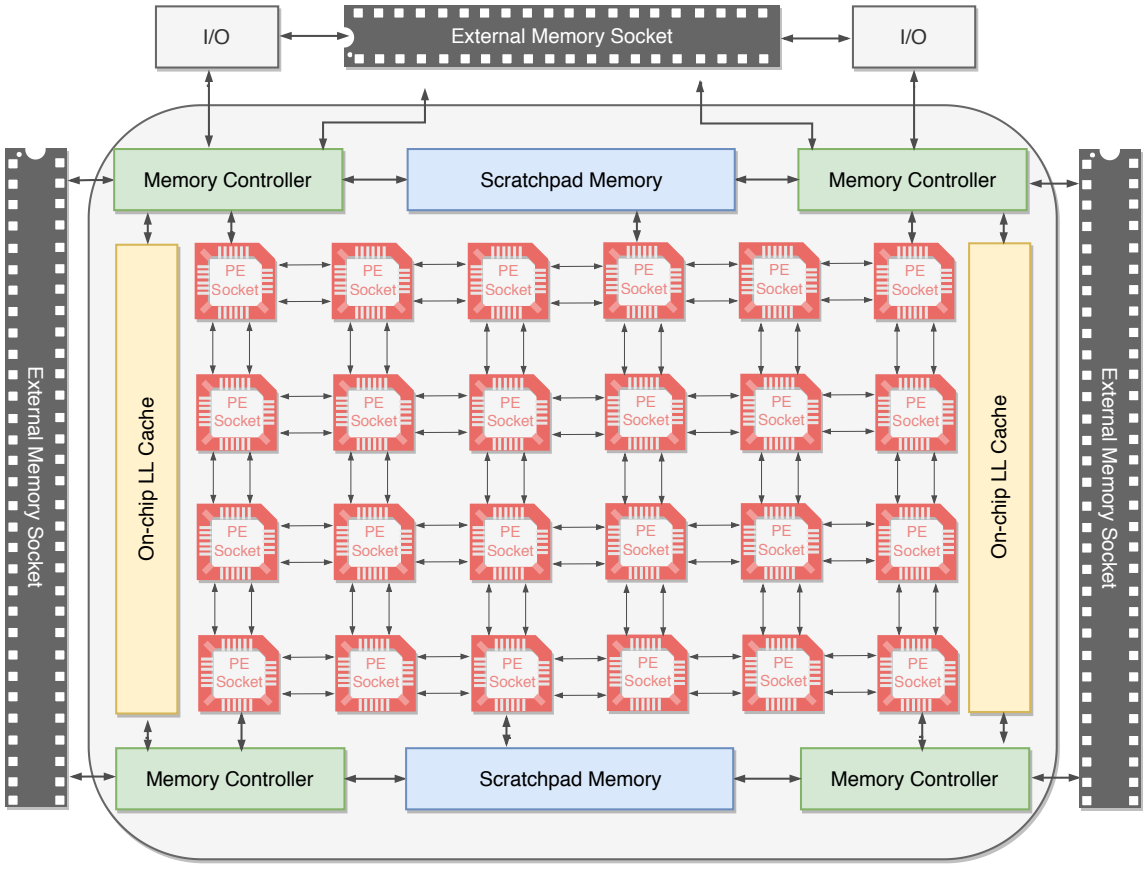


Figure 2·1: Nodes of different trust levels have their own local memory and cache. They are connected by an on-chip router.

The rest of the chapter is organized as follows: Section 2.2 briefly lists the related

work; Section 2.3 gives an overview of the security policy of the proposed design; Section 2.4 introduces the distributed key management of the architectural model, which is the core mechanism of the proposed system; Section 2.5 proposes 4 protocols to enhance the security of the architectural model on dishonest PE tolerance and PE privacy preserving; Section 2.6 explains the architecture support; and finally Section 2.7 concludes the chapter.

2.2 Related Work

Many hardware-based security techniques have been proposed in recent years (Wang et al., 2008). For most multicore system-on-chip, a secure core at each node is unnecessary. In these systems, mixed criticality or multi-tenant/multi-trust computation is often the design choice (Oberg et al., 2013). Different solutions have been proposed at different design abstraction levels, from gate-level description (Tiwari et al., 2009) to system virtualization (Hwang et al., 2008). Wassel et al. implemented a non-interfering scheme for secure NoC in SurfNoC (Wassel et al., 2013). Sajeesh and Kapoor (Sajeesh and Kapoor, 2011) highlighted some of the advantages of implementing security policies at the network interface level in NoC based systems for secure communication among such IP cores. Porquet et al. (Porquet et al., 2011) introduced a solution for co-hosting different protection domains or compartments on the same shared memory multiprocessor SoC using a NoC architecture. The proposed design model addresses both the hardware and software components of multi-tenant execution. It allows system designers to define and enforce execution communication rules for both secure and non-secure cores or software at the on-chip communication layer. Previously proposed secure processors (Chhabra et al., 2010) are still supported in the proposed architecture model since the security protocols are not bound by the processor core type. The design of the group key management scheme is informed

by the model of attacks highlighted in (Katz and Shin, 2005). In this model, if a secure processor core is used at a processing site, the system designer can bypass the network interface security module. In such cases, the traffic coming from the processor is treated as non-secure communication from the point of view of the on-chip network security protocol. In (Katz and Shin, 2005), an Authenticated Key Exchange amongst a group is explored. Using group keys allows a message to be sent to multiple recipients without having to pay the cost of encrypting the data multiple times.

2.3 Security Policy

The proposed new architectural model supports secure multicore computing architectures. It reduces the system attack surface by creating a virtualization layer that isolates processes based on system and user defined trust levels and security policies.

2.3.1 Process Isolation via Hardware Virtualization

In current SoCs, dynamic scheduling of tasks to processing elements makes it difficult to reason about the runtime interactions between functions of different trust levels, especially in the absence of hardware-level support. This procedure often leads to ad-hoc execution modes where trusted or untrusted software could be running on both trusted or untrusted hardware. Figure 2·2 shows a set of applications with mixed security mapped onto mixed security hardware. The new architectural model achieves both hardware and software views of secure processing by grouping processors into physical zones called *wards* and virtual logical zones called *islands*.

First, the on-chip processing elements are divided into *wards* that are identified and formed at system integration time, based on IP or processing element provenance. *Wards* are created to help negotiate the security keys used to create the *islands* in a trusted manner. Because the security level is inherent to the IP origin, the *ward* membership remains constant throughout the chip's lifetime. The chip is divided into

physical quadrants, and, within each quadrant, nodes of the same security level make up one *ward*. Hardware security is divided into highly trusted, trusted, untrusted, and unknown levels.

Each *ward* has a representative processing element/node, called the *anchor* node, specified and selected during system integration stage. The anchor node has a table containing the reachability and security information of the other *anchor* nodes and nodes in the same *ward*. Figure 2·3 shows the *ward* grouping of the illustrative mixed security heterogeneous architecture presented in Figure 2·2. The anchoring of *wards* provides a simple method of node discovery and key distribution without requiring a full list of node keys at each node.

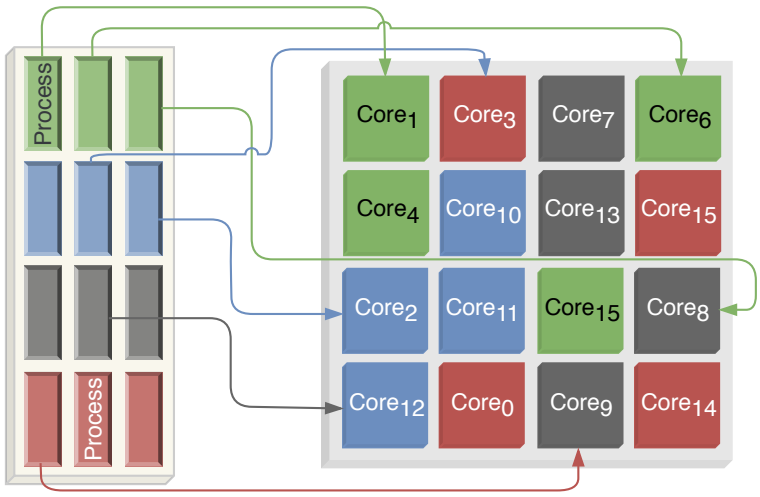


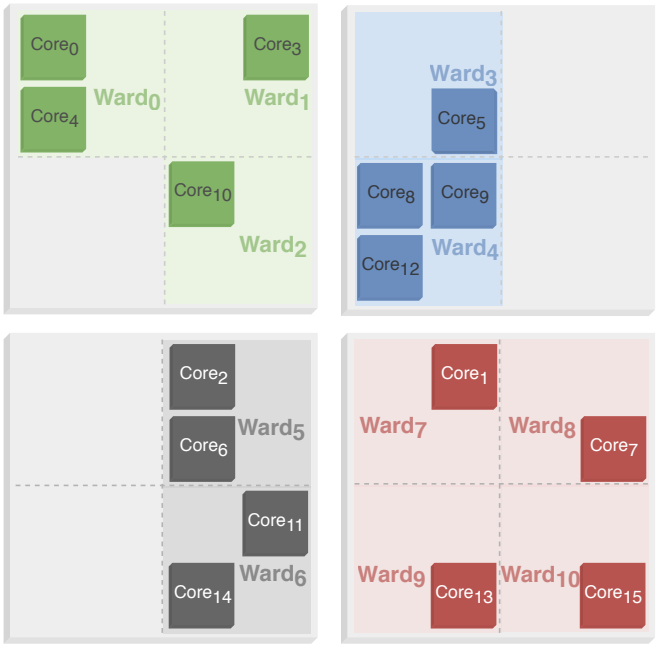
Figure 2·2: Trusted/untrusted applications running on trusted/untrusted cores. Different trust levels are illustrated by different colors (e.g., red represents the least trusted program or core).

Second, the processing nodes of the architectural model are virtually grouped into logical zones called *islands* based on their static or runtime security characteristics. Figure 2·4 shows an illustration of a virtually partitioned view of a multicore chip, based on each node's trust level.

Kernel and user applications are assigned a security abstraction, trusted or un-

trusted, that creates four basic island security levels namely: (1) trusted hardware and trusted software; (2) trusted hardware and untrusted software; (3) untrusted hardware and trusted software; and (4) untrusted hardware and untrusted software. Each node is assigned to an island based on the applications running on them and their IP trust credentials. Islands allow keys to be managed at runtime to maintain isolation as tasks are moved between the processing units.

One of the key innovations of the architectural model is that the physical *wards* and the virtual *islands* are decoupled, hence making the node placement decisions independent of the processing elements' security characteristics. This allows for efficient on-chip routing and node grouping.



Ward	Anchor	Members
0	$Core_0$	$Core_4$
1	$Core_3$	
2	$Core1_0$	
3	$Core_5$	
4	$Core_8$	$Core_9$ $Core_{12}$
5	$Core_2$	$Core_6$
6	$Core_{11}$	$Core_{14}$
7	$Core_1$	
8	$Core_7$	
9	$Core_{13}$	
10	$Core_{15}$	

Figure 2·3: Illustrative case of *ward* groupings with associated *ward* table.

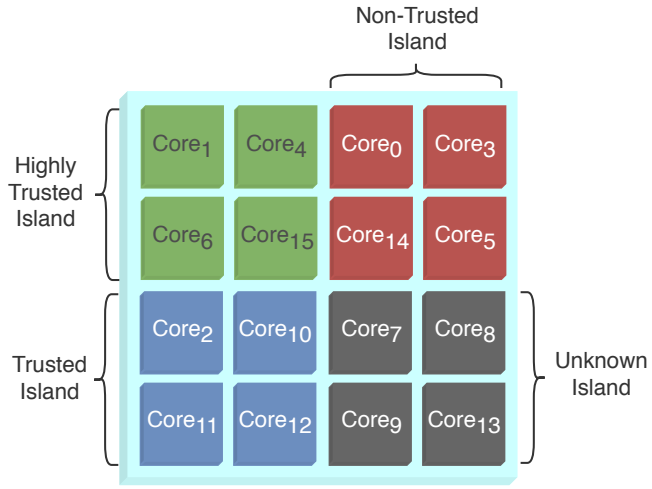


Figure 2·4: Hardware virtualization through highly trusted, trusted, untrusted and unknown island partitioning.

2.3.2 Enhanced Programmable Memory Access Management

In multicore systems, the implementation of on-chip security policies is typically handled by the memory management unit (MMU) through: (1) application memory management; (2) operating system memory management; and (3) hardware memory management. The MMU is placed between the processing element and the memory subsystem where it translates virtual addresses into physical addresses and performs access right validations. All of the memory management safeguards, however, stop at the processing node boundary.

The proposed design extends the MMU protections and security policies beyond the node boundary into the NoC layer while still applying conventional system memory management techniques at the node-level. The new architectural model is an interconnected network of nodes where the physical memory is distributed. A portion of the total on-chip memory is allocated to each processor node in a *Non-uniform Memory Access* (NUMA) style. Figure 2·5 shows the system with its enhanced node organization. The MMU not only provides local memory protection guarantees for

processing requests serviced at the node-level but also guarantees their secure routing when requests need to traverse the on-chip network by sending processor *id* and access code *AC* along with the messages.

On a ***load*** or ***store*** miss at the processing element, the higher order bits in the address are used to locate the physical location of the memory block being addressed. During packetization at the source, the processing element *id* is concatenated to the address and/or data with an access code (shorter version of island key). The packet is then encrypted using the island key or the destination master key, depending on whether the request is within or across islands. At the remote node, the security layer checks that if the process or processor making the request belongs to the appropriate island. After depacketization, the local router sends the source PE *id*, the address, the *access code* (AC), and the data if it is a write operation, to the memory module. The lower bits of the address are used to index into the *Access Code Table* (ACT) to check that the PE is a member of the island authorized to access the memory block. In parallel to this, memory boundaries and access code are fetched from *Base Table* (BT) to verify that the PE protection key matches the current access code value associated with the memory block being accessed. The AC, in addition to the Base and Limit registers, helps preserve forward and backward secrecy by being updated on changes in island membership, even over the same memory range. Figure 2·6 illustrates the memory access control logic.

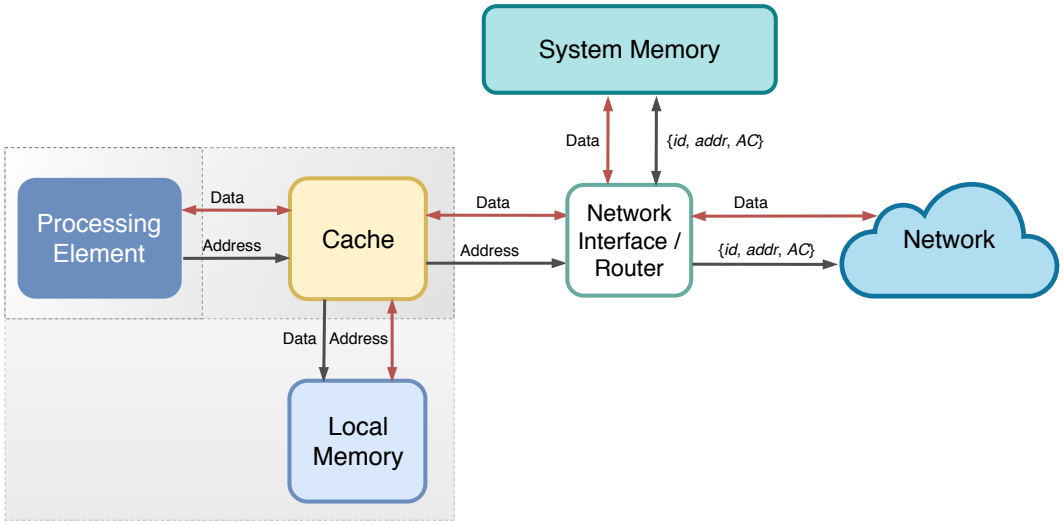


Figure 2·5: Programmable enhanced access control at the node boundary through the router network interface.

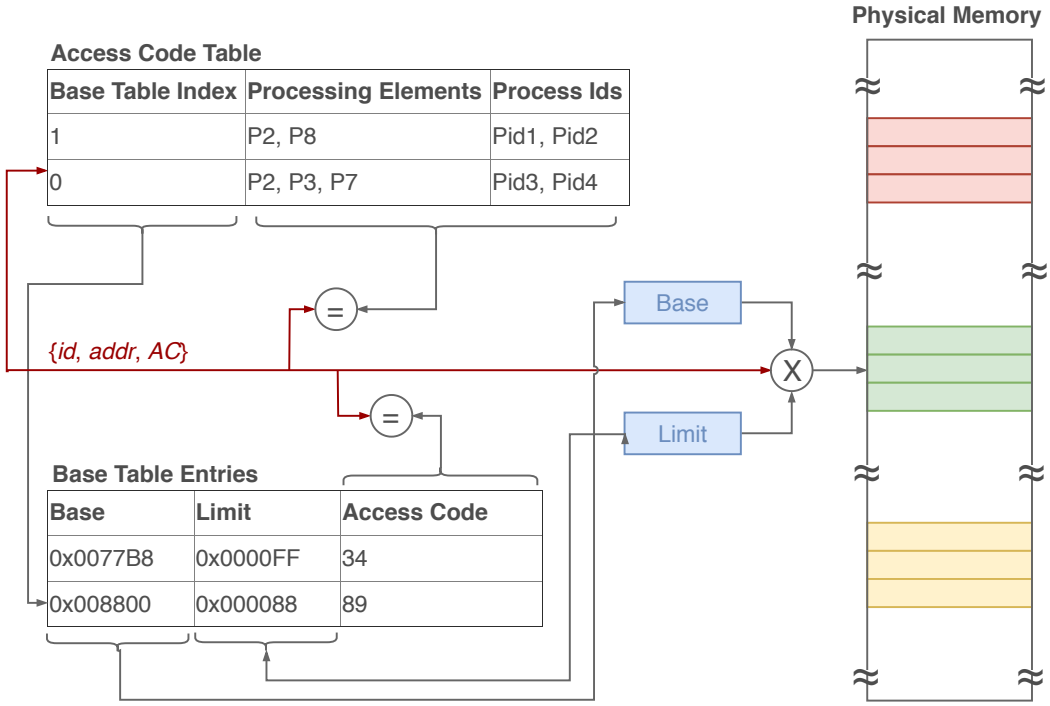


Figure 2·6: Access key managed memory zones.

2.4 Security Mechanisms: Distributed Island Key Management Approach

The proposed new architectural model uses an efficient dynamic key management protocol to manage and isolate various trust levels with low hardware-overhead (Kinsy et al., 2017). The protocol generates island keys based on the processing elements' and applications' trust levels.

For this type of distributed key management system, there are three general approaches for handling the distribution: (1) store the full list of public keys at each node; (2) store only neighbor's key at each node; and (3) divide the nodes into clusters, where one node in the cluster stores the public keys of the other clusters, while the remaining nodes in the cluster only store the lead node's public key. For practicality and security purposes, we implement the third approach: a distributed and coarse-grained method, where nodes only store certain public keys.

As the application data are placed onto the nodes, public keys are stored on *anchor* nodes. Each *anchor* node has a table containing the public key list of the other *anchor* nodes and the nodes in its *ward*. The notation for node keys is as follows: the public key of a node i is denoted K_{P_i}, the private key is K_{S_i}, and an island v's key is represented as K_{G_v}. Public requests and responses are denoted $rq_{K_{P_i}}$ and $rp_{K_{P_i}}$, respectively. The process of dynamically creating, expanding and contracting islands happens through the *join* and *leave* operations.

The *join* operation protocol is used to add a node to an island or to create an island. An island is a set of nodes with access to a particular data block. When a new node needs access to a data block, it first must obtain the public key of an island member (referred to as its sponsor) and join the island before requesting access. The sponsoring node will verify the security level of the requesting node and, if the security is sufficient, will initiate the key update process and provide the new node

with the island key. The full protocol is described below, assuming requesting node i is in anchor node x's ward and the sponsor node j is in anchor node y's ward.

Protocol 2.4.1. The protocol of a new node conducting the *join operation* is as follows:

1. Node i sends an encrypted message to its ward's *anchor* node x requesting node j's public key: $E_{K_{P_x}}(E_{K_{S_i}}(M_{ix}(rq_{K_{P_j}})))$. E and M denote the encryption operation and message, respectively.

2. *Anchor* node x sends an encrypted message to *anchor* node y requesting node j's public key on behalf of node i (denoted by n_i) including i's public key: $E_{K_{P_y}}(E_{K_{S_x}}(M_{xy}(rq_{K_{P_j}}, n_i, K_{P_i})))$.

3. *Anchor* node y sends an encrypted message to node i using i's public key containing node j's public key for node i: $E_{K_{P_i}}(E_{K_{S_y}}(M_{yi}(rp_{K_{P_j}})))$.

4. Node i then sends an encrypted message to node j using j's public key requesting to join the island assigned to the memory block at j: $E_{K_{P_j}}(E_{K_{S_i}}(M_{ij}(rq_{K_{G_v}})))$.

5. Node j verifies node i's access code embedded in the key request message to determine i's trust level. If there is no island, node j creates a symmetric key and sends it to i, $E_{K_{P_i}}(E_{K_{S_j}}(M_{ji}(rp_{K_{G_v}})))$. If there is an existing island, node j creates a new island key using a one-way function f such that $K_{G'_v} = f(K_{G_v})$. Node j sends the new island key $K_{G'_v}$ to both i and j sponsors for the island to enable the propagation of updates. Node j also marks its island table to reflect i as a dependent node. When sponsor key update reply comes back, then j sends i the new key.

6. When node i receives the message, it updates its island table to make j its sponsor for the particular island. □

Figure 2·7 (a) shows an illustration of the *join* operation. To reduce the number of messages for establishing or joining a new island, we add the message relay capability, where if two *anchor* nodes are at the same trust level, then the second *anchor* can directly send the island key request to the node in its *ward*, as shown Figure 2·7 (b).

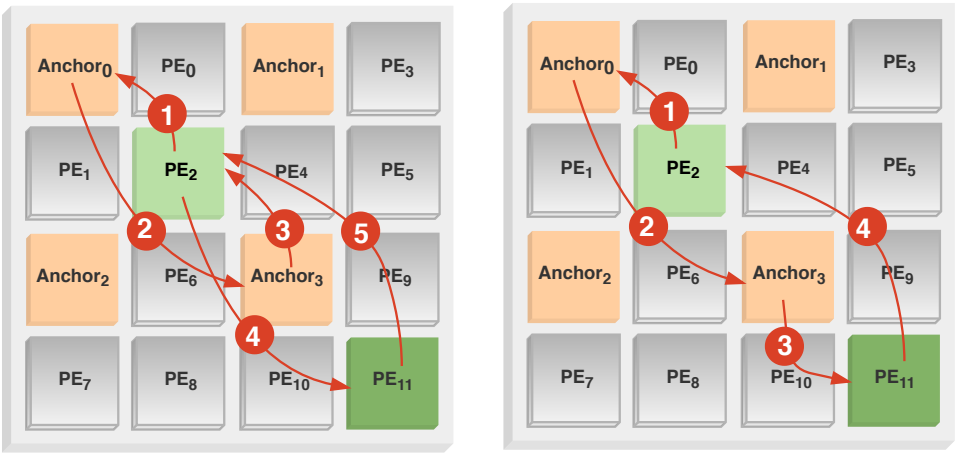


Figure 2·7: The two forms of the *join* operation protocol.

Protocol 2.4.2. The *leave operation* protocol is as follows:

1. Node i sends to node j, its sponsor for an island, a leave request. j sends the new island key $K_{G'_v}$ to its node sponsor of the island for propagated updates. When the sponsor's reply arrives, node j removes i from its island table and sends a reply to i.

2. When node i receives the message, it updates its island table for the particular island. □

Islands dynamically change when certain execution events occur. For example, when a cache-miss involves a remote access and the processing node making the cache request is not part of the *island* holding access key to the memory block of interest. This event will lead to an *island join operation*. Dynamic task and thread scheduling, re-scheduling and load re-balancing may also activate idle nodes and create *join operations*. When tasks or threads finish or exit the system, these events may trigger *leave operations*.

Both *join* and *leave operations* may lead to a new island topology. In such cases, the routing connectivity graph of the island needs to be rebuilt and associated routing tables will need to be updated. During the re-keying process, network virtual channels are also reset to prevent VC plowing.

2.5 Untrusted Processing Element Resistance and Privacy Preservation

Protocols 2.4.1 and 2.4.2 have provided a convenient and trustworthy approach to allow a node to *join* and *leave* a virtual *island*. However, in this approach, an assumption is made that, although nodes have different trust levels, they will remain honest in the entire *join* and *leave* processes. This means that: (1) a node i will only apply for an island using an access code AC matching its trust level; and (2) a *sponsor* node j will verify the requesting node i's trust level honestly and will not let in any of the disqualified nodes.

With any of the above conditions broken, there will be grave vulnerabilities in the system. For example, since the access code AC is a static string, any node who has knowledge of this code (by legal or illegal means) can request for an island key it does not deserve. Another scenario can be that a *sponsor* node j, say from an unknown island, treats its sponsorship with misconduct by letting in a node without a proper AC.

Additionally, there may also be a need for a requesting node to join and leave an island silently without being known by other nodes and even the anchor nodes, except a necessary *sponsor*. This applies particularly to the scenario when a highly trusted node has to join a lower trust-level island and it does not want to be listed as a potential *sponsor*. This invisible *join* ensures that its public key will not be requested by other untrusted nodes who also want to join the island.

In response to these vulnerabilities and privacy demands, we propose a set of corresponding solutions that enhance the *join* protocol 2.4.1. With these solutions:

1. We apply a dynamic access code AC fetching protocol, so that only the nodes properly being verified and applying for AC can present to its sponsor node a legal AC, which is also tailored for the requesting node.

2. We apply a threshold *join* authorization protocol, so that it takes more than one *sponsor* node to allow a *join* of a new node. Thus, even if one *sponsor* node accepts invalid *join* request, other *sponsor(s)* can prevent invalid joins.

3. We apply an invisible *island join* request protocol, so that a requesting node with a higher trust level can hide its lower trust-level destination from the anchor or any other nodes. The privacy of the node is preserved and it will not be listed as a potential *sponsor* in a lower trust-level island.

4. In addition, we propose a group anonymous authentication protocol (GAAP), which enables the new architectural model to verify the elements' trust levels without revealing their individual identities. This protocol can be applied to many area where authentication and privacy are both demanded.

This section describes the protections of the key management approach in Section 2.4, under the assumption of dishonest entities' existence. The protection scheme consists of 4 protocols to address the problems mentioned above.

2.5.1 Dishonest Requesting Nodes Resistance — Tailored Access Code (AC) Fetching

As introduced in Section 2.3.2, the access code AC is one of the key elements in the proposed architectural model. It is derived from the island key, and grants a node with the permission to access the block of data which matches the node's trust level. The AC is a static, shorter version of island key K_{G_v} and remains the same to any node acquiring it. Its generation and storage are not privacy preserving. This means any node with this string will be able to make a request, and there is no particular protection approach to prevent nodes from re-using it without authorization.

A desirable secure AC fetching would require that each node has to dynamically acquire a temporary (one-time) and tailored AC before its *join* request; this AC

should be valid only once for this request and for this node. When node i is authenticated and verified with its trust level, it needs to acquire a temporary AC, whose value is only known to itself. To do so, it will firstly send a masked string to the Access Code Table, who will sign the masked string. Then, i will remove the mask without destroying the signature. The signature and a proof of knowledge of the string will be sent to its prospective *sponsor* j for AC verification.

The figurative illustration of this procedure is shown in Figure 2·8.

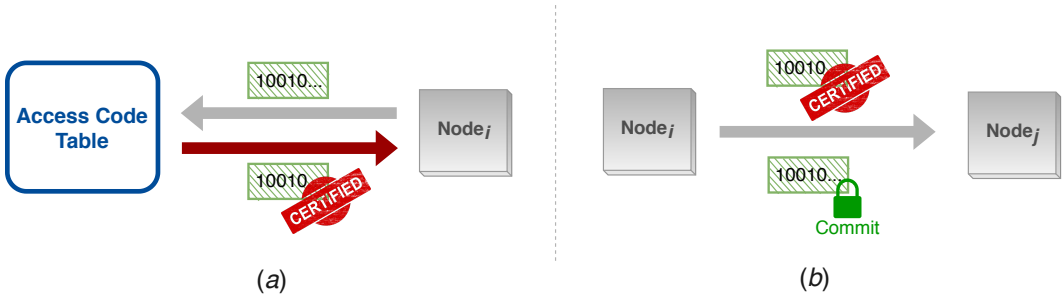


Figure 2·8: (a) A temporary AC is dynamically generated and signed. (b) The AC now becomes the signature and the proof of knowledge of the signed content, to be verified by node j.

Protocol 2.5.1. Inspired by the compact e-cash (Camenisch et al., 2005) scheme, we propose the following protocol for the dynamic AC fetching.

1. Node i firstly proves its authenticity and trust level to the Access Code Table (ACT) through authentication (e.g., through a hardware one-way function such as PUF (see Chapter 3)).

2. Node i selects an arbitrary string x, computes its commitment $COM(x)$. For simplicity, a commitment here can be considered as a one-way hash.

3. Node i selects a masking function $c()$, such that $c(x)$ leaks zero knowledge of x. In addition, there exists a function $c'()$ such that $c'(c(x)) = x$.

4. Node i shows $c(x)$ to the ACT. The ACT signs it with function $S'(c(x))$ and returns it to the device. In addition, there exists a function $S()$ that $S(S'(c(x))) = c(x)$. This function $S()$ is known by all the current island members.

5. At the time of the *join* request, node i applies $c'(S'(c(x))) = S'(x)$, which is the temporary AC, and sends $COM(x)$ together with $S'(x)$ to node j.

6. Node j applies $S(S'(x))$, and verifies if:

$$COM(S(S'(x))) \stackrel{?}{=} COM(x). \tag{2.1}$$

If so, then node j agrees to be the sponsor of node i. □

∗ (*Note*: The $c(), c'(), S(), S'()$ functions also satisfy commutative encryption property.)

In this way, the AC is dynamically generated and no longer a reusable string. Consequently, the *Access Code Table* (ACT) is replaced by the verification of equation $COM(S(S'(x))) \stackrel{?}{=} COM(x)$, which carries out the same task to check that the PE is part of the island authorized to access the memory block.

2.5.2 Dishonest Sponsor Tolerance—Threshold Join Authorization

In Protocol 2.4.1 Step 5, node j, the prospective *sponsor*, will need to first decrypt the request $E_{K_{P_j}}(E_{K_{S_i}}(M_{ij}(rq_{K_{G_v}})))$, and then check node i's AC. Upon the verification of the AC embedded in $rq_{K_{G_v}}$, node j determined whether to send the new island key $K_{G'_v}$ to node i.

Although in Section 2.5.1 the new model ensures the proper fetch of AC, if a *sponsor* node j is not honest, then it could let in any node with or without a qualified AC. This might not happen among the *sponsors* on the highly trusted islands or trusted islands. However, it is possible among the untrusted or unknown ones in Figure 2·4.

Therefore, we propose a dishonest sponsor tolerance scheme that does not put the trust on a single *sponsor*. Instead, it takes the permission from multiple *sponsors* to enable the *join* operation. In this scheme, we set a threshold t where node i must

acquire the permission from no less than t of any *sponsor* nodes in the desired island. Less than this threshold, node i is unable to join.

Such a group decision approach has been proposed by a number of research efforts (Ohkubo et al., 1999) (Iftene, 2007) (Chu and Tzeng, 2008). Briefly speaking, to gain the access to the desired island, node i must acquire an authorization token A. In that island, each of the existing nodes holds its own id and a ballot, which is computed by Shamir's secret sharing equation based on A and its id. After verifying the AC in node i's request message, the prospective *sponsors* who agree to grant i the island key will send their ballots to i, and those who do not agree will not. Node i needs to collect at least t ballots to retrieve A to prove its qualification for K_{G_v}. Figure 2·9 depicts the voting procedure.

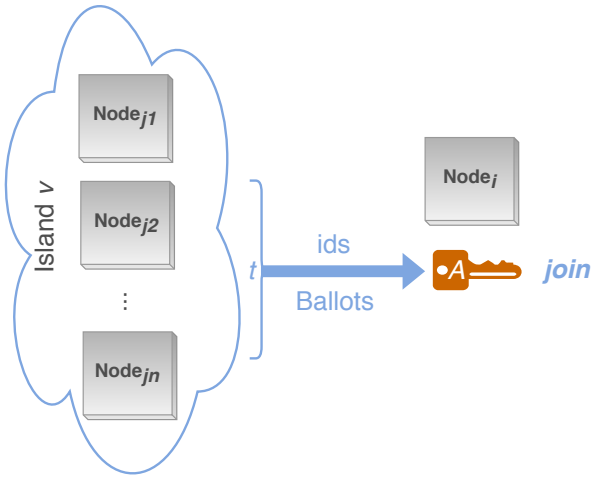


Figure 2·9: Node i has to collect at least t supportive *sponsors* to gain access to island v.

Protocol 2.5.2. The threshold island key authorization protocol with dishonest *sponsor* tolerance has two stages:

1. **Ballot Distribution:**

   (a) For an authorization token A at an island, its commitment $COM(A)$ is

made publicly known to all existing nodes;

(b) Denote node j's id as id_j, then node j's ballot β_j is computed by:

$$\beta_j = a_0 \oplus a_1 id_j \oplus a_2 id_j^2 \oplus \cdots \oplus A id_j^{t-1}, \tag{2.2}$$

where all the coefficients of a can be arbitrarily chosen.

2. **Threshold Voting:**

(a) Node i has to contact t prospective *sponsors*, who will verify its AC (some dishonest *sponsors* may not). Only the *sponsors* who agree to let i join will send their ballots to i. If there are at least t *sponsors* supporting the decision, then with the Lagrange interpolation formula:

$$A = \bigoplus_{j=0}^{t-1} \frac{\beta_j}{\prod_{j=0, j\neq k}^{t-1} (id_k \oplus id_j)}, \tag{2.3}$$

the authorization token A can be reconstructed by node i. If there are less than t supportive voters, then A remains unknown to i, meaning the *join* request is denied;

(b) Node i computes $COM(A)$ and proves to all the *sponsors* it reaches the qualification threshold. Then, the island key will be granted by any of the supportive *sponsors*.

(c) Once i joins the island, it will hold its ballot β_i as a potential *sponsor* too. □

Equations (2.2) and (2.3) are the share distribution and secret reconstruction equations of the t-threshold secret sharing (TSS) scheme, which was first introduced by Shamir (Shamir, 1979) and later studied by many researchers. With this technique, the probability of admitting a disqualified node with a single dishonest *sponsor* is eliminated. For example, at an unknown trust level island where $t = 2$, a dishonest *sponsor* decides to casually give its ballot to node i, whose AC is disqualified. Another *sponsor* checks the AC faithfully and decides not to support the *join* of i. Then, without enough ballots, information of A remains unknown to node i. Thus, i

cannot acquire K_{G_v} by showing $COM(A)$.

2.5.3 Threshold Join Authorization with Cheater Tolerance

The threshold authorization presented in Section 2.5.2 assumes that each sponsor will truthfully turn in their ballots (shares). It does not handle the situations when malicious sponsors (cheaters) distort their shares, or try to reveal the authorization token A collectively.

In this section we propose a stronger threshold join authorization scheme with cheater tolerance. We not only aim to detect any cheating in the authorization procedure, but also identify the cheaters.

Share Verification and Secret Verification

An interesting fact is that, the ballot/share distribution Eq. 2.2 is inherently equivalent to the non-systematic encoding equation of the well-known Reed-Solomon (RS) error correcting codes. RS codes are maximum distance separable (MDS) codes which meet the Singleton bound with equality. With such a distribution equation, an (n, t, d) Reed-Solomon codeword $(\beta_0, \beta_1, \cdots \beta_{n-1})$ is encoded with n symbols (shares) in total, t information symbols, and distance $d = n - t + 1$ which corrects up to $\frac{n-t}{2}$ erroneous symbols with algorithms in (Berlekamp, 2015), (Gao, 2003). Usually an assumption is made that there are no more than t dishonest shareholders so that they cannot recover the secret token. Denote the estimated number of cheating shareholders as c_{est}, then we have:

$$c_{est} < n/3 \tag{2.4}$$

[Eq. 2.4] indicates that if n instead of t shareholders are involved in the secret reconstruction and [Eq. 2.4] holds, then we can tolerate up to $n/3$ cheaters by a RS decoder while retrieving the correct secret. However, if the actual number of cheaters $c_{act} > c_{est}$, then there is a chance that the system will mis-detect cheaters.

Another approach is to verify the secret (Wang et al., 2016b). Usually before being shared, the secret is encoded with some modification detection capability. Then after reconstruction the system will be able to verify the secret's authenticity to tell whether there is any dishonesty. However, this approach usually can only detect but not identify the cheaters. A standard cryptographic hash function (HMAC) can be used here. An alternative is the Algebraic Manipulation Detection (AMD) code [(Wang and Karpovsky, 2011)] which supports flexible digest size.

Unlike HMAC, the AMD codes operate over finite fields and its security level is adjustable by block size b. The AMD encoding is defined as follows:

Definition 2.5.1. Let $K = (K_1, K_2, \cdots, K_m)$, where $K_i \in GF(2^b)$ is a randomly generated b-bit vector. An g^{th} order Generalized Reed-Muller code (GRM) with m variables consists of all codewords $(f(0), f(1), \cdots, f(2^{bm}-1))$, where $f(K)$ is a polynomial of $K = (K_1, K_2, \cdots, K_m)$ of degree up to g. Let

$$A(K) = \begin{cases} \bigoplus_{i=1}^{m} K_i^{g+2}, & \text{if } g \text{ is odd;} \\ \bigoplus_{i=2}^{m-1} K_1 K_i^{G+1}, & \text{if } g \text{ is even and } m > 1; \end{cases}$$

where $\bigoplus$ is the accumulated sum in $GF(2^b)$. Let

$$B(K,S) = \bigoplus_{1 \le j_1 + j_2 + \cdots + j_1 \le g+1} y_{j_1, j_2, \cdots, j_m} \prod_{i=1}^{m} K_i^{j_i},$$

where $\prod_{i=1}^{m} K_i^{j_i}$ is a monomial of R of a degree between 1 and $g+1$. And $\prod_{i=1}^{m} K_i^{j_i} \notin \triangle B(K,S)$ which is defined by:

$$\begin{cases} \{K_1^{h+1}, K_2^{g+1}, \cdots, K_m^{g+1}\}, \text{if } g \text{ is odd;} \\ \{K_2^{g+1}, K_1 K_2^{g}, \cdots, K_1 K_m^{g}\}, \text{if } g \text{ is even and } m > 1. \end{cases}$$

Let $f(K,S) = A(K) \oplus B(K,S)$, then a generalized AMD codeword is composed of the vectors $(S, K, f(K,S))$, where S is the information portion, K the random vector, and $f(K,S)$ the redundancy signature portion [(Wang and Karpovsky, 2011)]. ■

Remark 2.5.1. If the attack involves a non-zero error on the information S, which is the major purpose of almost all attacks, then in $f(K,S)$ the term $A(K)$ can be

omitted [(Bu and Karpovsky, 2017)]. Further more, if only one random number vector is used, the encoding equation can be further more simplified to:

$$AMD(K,S) = f(K,S) = \bigoplus_{1 \le j_1 + \cdots + j_i + \cdots + j_m \le h+1} S_{j_1,\cdots,j_i,\cdots,j_m} K^{j_i} \tag{2.5}$$

where S_{j_i} is a b-bit block of S. ■

The Robust and Adaptive Secure Secret Sharing Scheme (RASSS)

In this section we combine the share and secret verifications with group testing and hardware root of trust, to construct a secure and robust secret sharing scheme (RASSS) with strong cheater tolerance (Bu et al., 2018c). RASSS has 4 adaptive stages, that a more powerful stage will only be activated when the previous stage fails. Thus the scheme functions in a cost-efficient way and consumes minimal resources on average.

The work flow of the proposed scheme is shown in Fig. 2·10.

Stage 1: Encoding and Distribution of the Secret

First, we setup a authorization token reconstruction hardware module. This module takes shares and IDs from sponsors and performs [Eq. 2.3] to reconstruct the token. Also it is equipped with a secure key sharing channel on it. One trustworthy and lightweight solution is to use a physical unclonable function (PUF), whose challenge-response pairs can be used to establish key agreement.

Before sharing the secret authorization token, the proposed architectural model will encode the token A with an Encryption-then-MAC function $EtM()$ to $E = EtM(K, A)$. K is a PUF response randomly picked from the dealer's repository, which stores the challenge and response pairs (CRPs) of the reconstructor's PUF. The MAC function is for secret verification, and the encryption function is to prevent passive cheaters to retrieve A stealthily. Let CHL be the corresponding challenge to K. Then the dealer shares E to all shareholders with the following new distribution

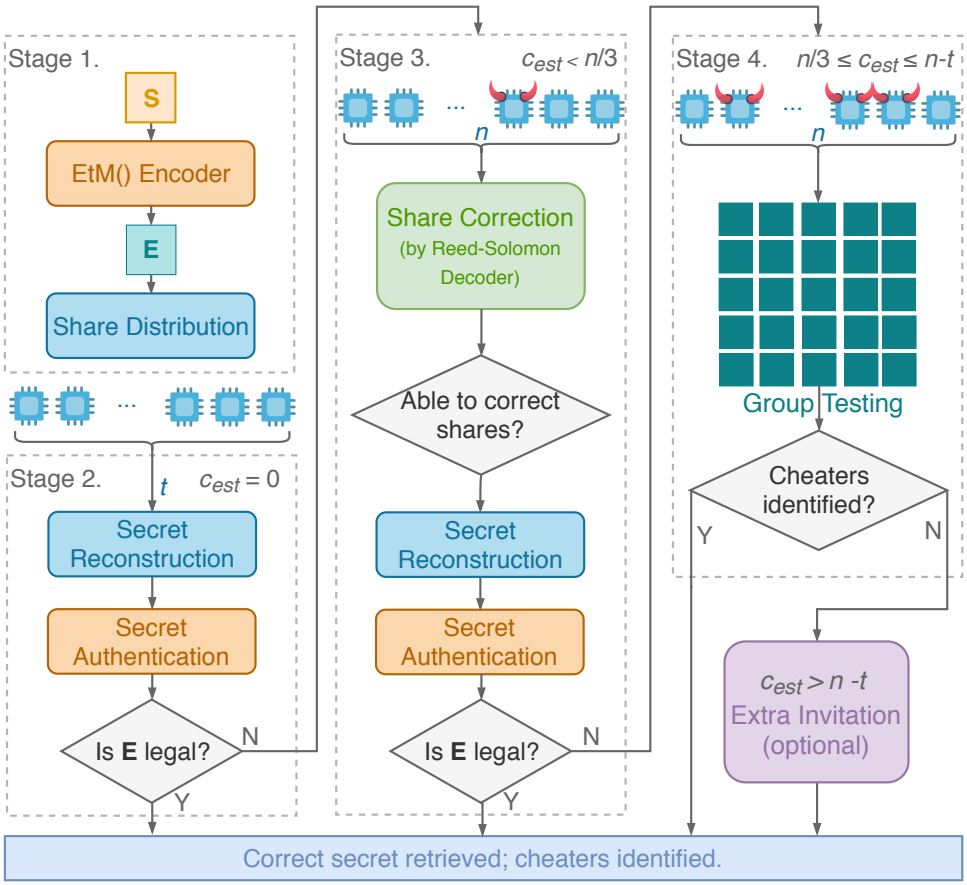


Figure 2·10: Stage 1 and **2** are sufficient if the number of actual cheaters $c_{act} = 0$. If cheating is detected by **Stage 2**, then **Stage 3** with RS decoder is called under the assumption of $c_{est} < n/3$. If **Stage 3** fails then **Stage 4** with group testing is able to identify $n/3 \le c_{est} \le n - t$ cheaters. If c_{act} is even beyond this scale, an extra invitation module can be introduced to resolve the issue.

equation (Bu et al., 2017):

$$\beta_i = CHL \oplus a_1 id_i \oplus a_2 id_i^2 \oplus \cdots \oplus E id_i^{t-1}. \tag{2.6}$$

Stage 2: Secret Reconstruction

At the voting/authorization moment, first there will be t shareholders to participate in the secret retrieving. The reconstructor will use [Eq. 2.3] to retrieve $\tilde{E}$, and the

following Lagrange interpolation formula to retrieve CHL:

$$CHL = \bigoplus_{i=0}^{t-1} \frac{id_i \cdot \beta_i}{\prod_{j=0, j\neq i}^{t-1} (id_i \oplus id_j)}. \tag{2.7}$$

The reconstructor now applies CHL to its own PUF to acquire K. Now it is able to authenticate $\tilde{E}$ by the message authentication code in the $EtM()$ function.

If the authentication claims validity of the secret, then it is considered a successful secret reconstruction with no cheat. If not, the scheme calls for Stage 3 for share correction.

Stage 3: Client - Share Error Correction

This stage uses the Reed-Solomon error correction module in the classic protocol. Here, $n = 3c_{est} + 1$ shareholders will be invited to participate in the protocol, where c_{est} is the number of estimated cheaters defined by the system. The RS decoder will try to correct the shares and then send them back to the secret reconstructor. If it passes both the share correction (by RS decoder module) and secret verification (by authentication function), then the secret reconstruction is successful. When $c_{act} < n/3$, the cheater tolerance probability is 100%. If either module fails then the protocol ascends to its fourth stage, indicating that the actual number of cheaters $c_{act} \geq n/3$.

Stage 4: Client - Group Testing

This stage will be activated if the previously retrieved secret is not legal. It will involve up to n shareholders, among whom there are at least $n/3$ cheaters. The client will generate a group testing pattern which is able to identify up to $c_{est} = n - t$ cheaters with a minimum number of t honest holders. Even if there are more than $n-t$ cheaters, it is still able to detect the cheating by the message authentication code (MAC), although the correct secret is beyond reconstruction because of insufficient honest holders.

The group testing pattern's construction is as follows.

Construction 2.5.1. For any t-threshold secret sharing scheme, suppose among n holders there are c_{est} cheaters where $0 \leq c_{est} \leq n - t$. A test pattern to identify the honest holders and attackers can be constructed as a binary matrix M of size $T \times n$, where T is the number of tests needed at most. The rows of M consist of all different n-bit vectors with exactly t 1's and so $T = \binom{n}{t}$. Each column of the matrix therefore has $\binom{n-1}{t-1}$ number of 1's. The 1's in each row (test) correspond to the shareholders participating in that particular test. Each test is a two-step procedure:

1. A secret reconstruction to retrieve the secret $\tilde{E}$ with its specific participants;
2. An authentication to retrieve the challenge CHL and K to verify the validity of $\widetilde{E}$.

The test syndrome is a T-bit binary vector u, where 0's in u indicate a pass, and 1's authentication failure. □

The cheater identification algorithm is then described in Algorithm 1.

Algorithm 1: Cheater Identification Algorithm

For any t-threshold secret sharing scheme and its corresponding group testing matrix M there are n shareholders participating in the tests indexed by $H = \{0, 1, 2, \cdots, n-1\}$. Among the n shareholders there are c_{est} cheaters where $n/3 \leq c_{est} \leq n - t$.

1 Let $w = (w_0, w_1, \cdots, w_{n-1})$ be a n-digit vector and $w = u^\top \times M$, where u is the T-bit binary test syndrome and $\times$ is the multiplication of regular arithmetic.

2 The cheaters' indexes belong to the set $\{l|\ w_l = \binom{n-1}{t-1}\}$. and the rest of the holders are honest. □

As we can see, the testing technique in Algorithm 1 requires $\binom{n}{t}$ tests in total to identify the cheaters. This can be a large number when n and t are large. Therefore its adaptive form is given below in Algorithm 2 which drastically reduces the average number of tests to a linear function.

If the group testing module in Stage 4 cannot successfully identify the c_{act} cheaters in the system, where $n - t < c_{act} \leq n$, then the number of honest shareholders is less than t.

At this point, the proposed scheme will still raise the cheating alarm based on the secret authentication. Moreover, the protocol is adaptive enough to be extended

Algorithm 2: Adaptive Cheater Identification Algorithm

For a test pattern M of size $T \times n$ generated by Construction 2.5.1. Let $\triangle T$ be the number of tests needed to find the first 0 (a pass of authentication) in the test syndrome.

1. The n shareholders are indexed by $H = \{0, 1, 2, \cdots, n-1\}$. The t honest holders identified by this test are indexed by $I = \{i_0, i_1, \cdots, i_{t-1}\}$.
2. Now the system only needs to run at most $n - t$ more tests whose participants are $\{i_0, i_1, \cdots, i_{t-2}, j\}$, where $j \in H \backslash I$. Each test's syndrome indicates holder j as an attacker or not by 1 or 0.
3. The total number of tests needed to identify all holders is then at most $\triangle T + (n - t)$. □

to a further stage to include an invitation module. This module can pull in the execution additional participants and perform new rounds of group testing, as shown in Algorithm 3. From the hardware perspective, the invitation module can be power-gated and disabled when not in use.

Algorithm 3: Extra Invitation Algorithm

Let the number of honest shareholders in the current group testing be $\triangle t$ and $0 \leq \triangle t < t$.

1. Suppose the system is able to identify an extra set of t honest shareholders from another group. Then these t honest parties can be combined into the current group with the modified group testing matrix of size $\binom{n+t}{t} \times (n + t)$.
2. With this new test pattern, the $\triangle t + t$ honest shareholders can be identified and the rest will be properly labeled as cheaters. □

2.5.4 Invisible Island Join

In these heterogeneous SoC architectures, the attacker needs to first identify the victim node before devising an attack scheme. Consequently, any degree of node identity obfuscation will harden the system's security posture. Concretely, in cases where a highly trusted node i wishes to join an untrusted or unknown island, it may be favorable if its privacy can be preserved when communicating with other lower trust-level nodes and even the anchor nodes who have introduced i to its sponsor j. In a sense, node i enters an "incognito" mode.

For security purposes, we only allow trusted and highly trusted nodes to have this invisible island *join* feature. This feature allows a node i to hide its destination from

the *anchor* who introduces it to the *sponsor* j, while still getting the public key of j from the *anchor*, so that in the entire network, only j knows the *join* of i.

The brief idea is to make the public key request in an oblivious manner. In Protocol 2.4.1 Step 1, instead of putting the $rq_{K_{P_j}}$ in the request message, i uses an obfuscated message which does not reveal j's identity. In step 3, *anchor* y responds also with obfuscated public keys of all the nodes in its ward. Then, i is only able to retrieve j's public key but not others.

Figure 2·11 depicts the invisible join procedure:

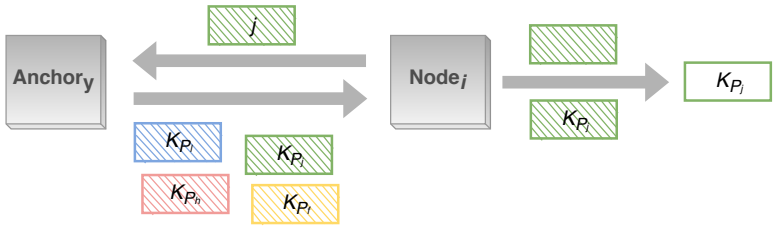


Figure 2·11: When node i makes an obfuscated request for j's public key, *anchor* y returns to i all the obfuscated public keys in its ward. The obfuscation approach of those public keys by y is related to i's request, in a way that all the public keys are masked differently, and only j's key can be recovered by i.

Protocol 2.5.3. The invisible *join* protocol is as follows:

1. When node i wants to have j as its *sponsor*, it firstly notifies *anchor* y through *anchor* x that it requests for one of the public keys *anchor* y holds;

2. Suppose *anchor* y has m nodes in its ward. It generates m random vectors $\{r_0, r_1, \cdots, r_l, \cdots, r_{m-1}\}$ with the same length of the public keys;

3. *Anchor* y has a pair of public-private keys $\{pk_y, sk_y\}$. *Anchor* y sends pk_y together with the m random numbers to i, while keeping sk_y to itself privately;

4. Particularly, i wants to connect with j, so i generates the following vector:

$$u = r_j + Enc_{pk_y}(T), \tag{2.8}$$

where T is a random vector and $Enc()$ is a public-key encryption function;

5. On receiving u through *anchor* x, *anchor* y calculates:

$$t_l = Dec_{sk_y}(u - r_l) \tag{2.9}$$

for all $l \in \{0, 1, \cdots, m-1\}$, and $Dec()$ is a public-key decryption function.

6. *Anchor* y sends all these m messages to i:

$$s_l = t_l + K_{P_l}, \tag{2.10}$$

for all $l \in \{0, 1, \cdots, m-1\}$.

7. Node i computes:

$$K_{P_j} = s_j - T. \tag{2.11}$$

8. In this way, node i acquires the public key of node j in the desired island. All other public keys remain unknown to i, and the identity of j is unknown to all other nodes including the *anchor* nodes, except j itself. □

This protocol leverages the 1-out-of-n oblivious transfer (OT) (Tzeng, 2004) (Li et al., 2005) which provides invisibility of node i's *join* into j's island. This way i stays unlisted from the *sponsor* list and even *anchor* nodes x and y in the join Protocol 2.4.1 do not know where it has joined. Thus, i's public key will not be requested by other low trust level nodes who want to join the island.

2.5.5 Group Anonymous Authentication Protocol

Although we leverage an oblivious transfer-assisted invisible *join* protocol to enable elements with anonymity, their true identities are still revealed in each of the authentication procedure. The authentication is a critical procedure and cannot be omitted, since it is the foundation on which the the proposed architectural model determines the security level of each element.

However, an ideal privacy-preserving system would be able to verify the elements' attribute (security level, privilege etc.), without revealing the elements' identities. We

now introduce a novel Group Anonymous Authentication Protocol (GAAP), which enables the proposed architectural model to: (1) authenticate each element's security level (trusted, untrusted, unknown), (2) without compromising the element's anonymity.

In this protocol, each element still needs to perform authentication (through a hardware one-way function, e.g., PUF (see. Chapter 3)) to verify its legitimacy in the very beginning. Meanwhile, the verifier (the proposed architectural model) will prepare for each trust level a unique pool of authentication tags. Then, according to their trust levels, elements will fetch their own authentication tags from the pools in a double-blind manner, which will be used for their authentication later on. The tag fetching must satisfy a set of functionality and security criteria as follows.

Definition 2.5.2. The **functionality** of GAAP should satisfy:

(i) *Anonymity*: All elements in the same security level (group) should be indistinguishable to the verifier;

(ii) *Unlinkability*: The verifier should not be able to link several authentication requests/tags to the same anonymous element;

(iii) *Group Attribute*: The verifier should be able to distinguish requests from different groups (security levels) by matching the authentication tags, without revealing the individual identities of the requesting elements;

(iv) *Collision-free Tag Fetching*: When the tags are fetched in a double-blind manner, the verifier should be able to identify any fetch collision (two or more elements about to fetch the same tag) without revoking the anonymity;

(v) *Multi-Group Membership*: An element should be able to belong to multiple groups. In other words, groups can have overlaps.

Definition 2.5.3. The **security** of GAAP should satisfy:

(i) *Curious Authority-resistance*: A curious authority such as a verifier, a key distributor, or a certificate manager, should not be able to learn the individual identity of any element;

(ii) *Impostor-resistance*: Any element in a group should not be able to spoof another. In addition, any number of collusive dishonest elements should not be able to acquire more information than what they are granted;

(iii) *Eavesdrop-resistance*: A passive Man-in-the-Middle (MITM) should not be able to acquire any information by eavesdropping the channel.

Before the mathematical definition, a figurative analogy is presented to illustrate its major principles.

Analogy 2.5.1. A manager wants to give several groups of employees some one-time passcodes to different buildings. So that each employee in a certain group can access a corresponding building anonymously. For example, in group A with three employees, the manager prepared three different passcodes for building x. Now the passcodes should be picked up in a way that:

a. For security reasons, the passcodes cannot be left in an unsupervised room for the employees to pick up freely;

b. There should be no collision in the picking of the codes among the three employees;

c. The manager should not know any employee's code selection;

d. An employee should not know his two colleagues' codes.

Thus, any employee can enter building x with his/her passcode, which does not leak any information on his/her personal identity.

Such a demand can be implemented with the procedure illustrated in Fig. 2·12.

(1) Initial setup:

(a) There are three types of papers $(blue, green, red)$ and three corresponding solutions $(\alpha_b, \alpha_g, \alpha_r)$. When a solution is applied to its pairing type of paper, it does no harm to it. However, when it is applied to other types, it permanently wipes out all the content from them. passcodes a, b, c are written to the $blue, green$, and red papers respectively for multiple copies;

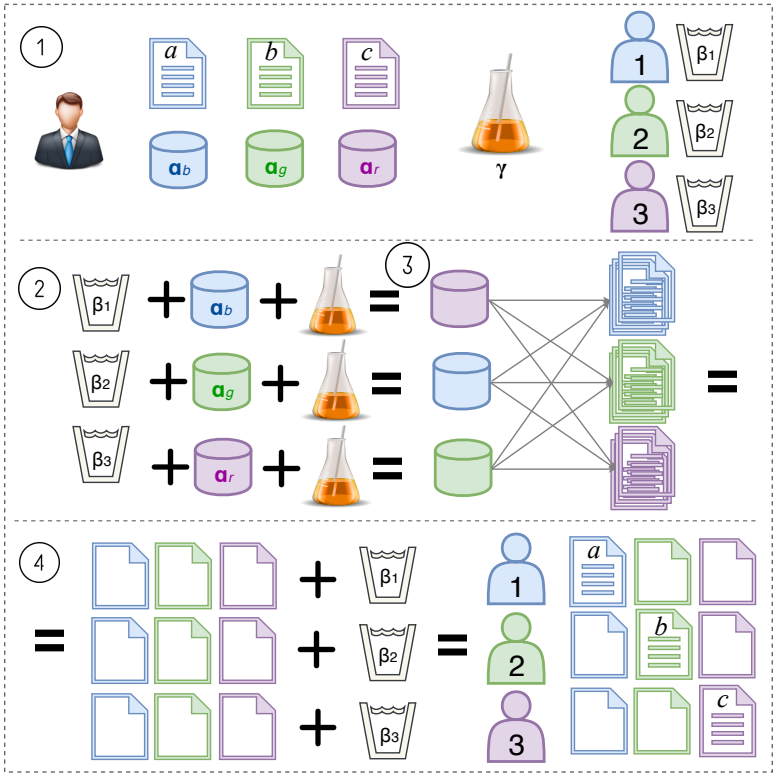


Figure 2·12: A 4-step passcode blind-fetching protocol. With this protocol, each employee can only acquire his/her own passcode, and will remain completely unaware of other codes. The manager also does not know the code selection of the employees.

(b) In addition, each of the 3 employees has a special colorless solution denoted by $\beta_1, \beta_2, \beta_3$, which can temporarily erase the content from any type of papers, and by applying this solution again the content will be restored. But β_i cannot restore the content masked by β_j if $i \neq j$.

(c) At last, there is a solution named γ, that when mixed with α it changes its color with a random bijection mapping back in $\{blue, green, red\}$, which is unknown to the manager.

(2) The three employees are given three different solutions, and they come to an agreement that each one picks one solution. At this point zero information of the passcodes is known to anyone. In Fig. 2·12 employee 1 picks α_b in order to acquire passcode a, employee 2 picks α_g to acquire passcode b, and employee 3 gets α_r for code c. Their own solution β is mixed with the selected α respectively, and then with solution γ. The resulted solution colors are a bijection of the

original color set. For example, in Fig. 2·12, $\alpha_b + \beta_1$ is converted to red, $\alpha_g + \beta_2$ to blue, and $\alpha_r + \beta_3$ to green.

(3) The manager looks at the three new solutions and has no idea of what their original colors are. Thus he also does not know about their passcode selection. However, the manager can tell that they have made a collision-free selection. So he applies each of the solutions to a set of passcode papers $(green, blue, red)$ with a, b, c on them. All the papers are now wiped out, some temporarily, and some permanently.

(4) When each employee gets back his/her set of blank papers, he/she applies his/her own β solution to those papers. Only the passcode on the previously picked color paper can be restored, and the other two papers are garbage. The employee can now use his own code (employee 1 gets a, 2 with b, 3 with c) to access building x without being distinguished. □

The GAAP Protocol

In this protocol, an element (in distributed systems, devices) first proves its legitimacy by being authenticated through its PUF. At this stage there is no anonymity and the purpose is only to validate the participants for the tag fetching later on. Then, the elements acquires their own authentication tags in a double-blind but verifiable manner according to their security levels. Through tag matching, the elements can be authenticated anonymously with only their security levels revealed.

We first define the following functions. The definitions are brief, so that the focus remains on the protocol. More of the functions' details will be given later.

- **TagGen(**n, m**)**: for a group of n elements where each element can be authenticated m times, the verifier uses this function to generate nm tags.

- **IndexSel(**$\{j\}, \{i\}$**)**: all elements in a group use this function to create a bijection mapping from the element IDs to the tag indexes. The output of this function is a set $\{c_j\}$, where $c_j = i$, meaning element j has chosen the i^{th} tag.

- **CoInChk(**$\{z\}$**)**: this function checks if there are duplicates in the extensional set $\{z\}$.
- **Enc(**x, pk_j**)**: the verifier uses this function to encrypt a message x with the public key pk_j of element j.
- **Dec(**y, sk_j**)**: element j uses this function to decrypt y with its private key sk_j.
- Initializer Module (IM): its only task is to generate and distribute random numbers. It has no other functional capability and cannot participate in any other activity.
- $\oplus$: an involution operator such as xor.

Protocol 2.5.4. Let us assume that there are w elements and u groups (e.g., partitioned by security levels) in a system. Each element is firstly authenticated through its PUF with its identity revealed to the verifier. For an arbitrary group g, there are n elements. The Group Anonymous Authentication for group g is performed as follows – other groups operate in a similar manner:

1. For the group g, the verifier uses **TagGen(**n, m**)** to generate nm arbitrary authentication tags t_i stored in set T_g. There are n elements $\{Elm_j\}$ indexed by $j \in \{0, 1, \cdots, n-1\}$. Each element fetches m tags to enable m authentication sessions. Every element has a public-private key pair: $\{pk_j, sk_j\}$;
2. In the 1^{st} round, the verifier arbitrarily takes n tags out of the nm tags, and indexes them by $i \in \{0, 1, 2, \cdots, n-1\}$ as $\{t_0, t_1, t_2, \cdots, t_i, \cdots, t_{n-1}\}$;
3. The n elements come to an agreement over the index i selection using the function **IndexSel(**$\{j\}$**,** $\{i\}$**)** $= \{c_j\}$. Each element's choice is $c_j = i$, meaning that the element Elm_j plans to acquire t_{c_j}. This agreement should be performed in a collision-free manner. Any double-dealing in the agreement will be spotted in step 6) by the verifier;
4. A Initializer Module (IM), which essentially is a random number generator, generates n random vectors $\{r_0, r_1, r_2, \cdots, r_{n-1}\}$, and a random number $d \in \{0, 1, 2, \cdots, n-1\}$. The IM sends d and r_d to all the elements. It also sends

all the random vectors $\{r_0, r_1, r_2, \cdots, r_{n-1}\}$ to the verifier. The transmission can be protected against MITM eavesdropping simply by using the public key systems of the verifier and elements;

5. Each element computes
$$e_j = c_j + d \pmod{n}, \tag{2.12}$$
and sends e_j to the verifier. e_j leaks zero-knowledge of c_j since the verifier has no knowledge of d;

6. The verifier checks if there are any index selection collisions using the function **CoInChk(**$\{e_j\}$**)**, where $j \in \{0, 1, \cdots, n-1\}$. If any element fools the index selection agreement function **IndexSel()** in step (3), it will be detected with probability of 1 without nullifying the tag selection process. If no collision, the verifier proceeds;

7. For any element Elm_j's e_j, the verifier computes:
$$f_i = t_i \oplus r_{e_j - i \pmod{n}} \tag{2.13}$$
for all $i \in \{0, 1, \cdots, n-1\}$, and stores the results in the set of $\{f_i\}_{Elm_j}$ whose cardinality is n;

8. The verifier uses each element's public key pk_j to compute **Enc(**$\{f_i\}_{Elm_j}, pk_j$**)**, and sends the results to Elm_j;

9. When element Elm_j receives its tag set, it uses **Dec(Enc(**$\{f_i\}_{Elm_j}$**,** pk_j**),** sk_j**)** to decrypt and retrieve $\{f_i\}$. Then by the previously received r_d from the IM, Elm_j computes its selected authentication tag by:
$$t_{c_j} = f_{c_j} \oplus r_d, \tag{2.14}$$
and it has zero-knowledge of the other tags;

10. All the participants repeat the steps above for another $(m-1)$ rounds allowing each element to acquire m tags. In each round, the IM generates a new set of random vectors $\{r_0, r_1, r_2, \cdots, r_{n-1}\}$ and a new d;

11. During the authentication itself, an element shows one of its tags to the verifier to prove that it belongs to group g. If the shown tag matches with a tag in the verifier's set T_g, the element is successfully authenticated. Then the element

can legitimately request privileges assigned to its group without revealing its individual identity. □

Remark 2.5.2. In Protocol 2.5.4 steps 3 and 4, c_j is equivalent to the solution α in the "building passcode" illustration presented in the beginning of this section. The d generated by IM is equivalent to the solution γ, which obfuscates the passcode selection to the verifier. Each device's public-private key pair is equivalent to the solution β.

Protocol 2.5.4 step 7 [Eq. 2.13] is equivalent to the step (3) in the "building passcode" illustration, which erases the selected paper temporarily, and destroys all other papers permanently. It ensures the double blindness that the verifier does not know which t_{c_j} is revealed to Em_j, and Elm_j does not know other elements' tags neither.

We now prove the GAAP's satisfaction on Definition 2.5.2 and 2.5.3.

Proof: Satisfaction of functionality Definition 2.5.2

(i) *Anonymity*: Since the selection of authentication tags is obfuscated by d which is unknown to the verifier, the verifier does not know the obfuscated index selection;

(ii) *Unlinkability*: Since in Protocol 2.5.4 step 10, during each round of tag fetching a new obfuscation vector d is generated, the verifier will not be able to link two tags fetched by the same element in two independent rounds;

(iii) *Group Attribute*: By generating unique pools (sets) of authentication tags for different groups, the verifier is able to distinguish the group attribute of all the devices;

(iv) *Collision-free Tag Fetching*: Although d is unknown to the verifier, this unique offset creates a bijection mapping from the index selection to the obfuscated index selection. Therefore any tag fetching collision can be immediately spotted;

(v) *Multi-Group Membership*: An element can fetch multiple groups' authentication tags in order to be a member of them, as long it is properly authenticated in the very beginning through its PUF. ■

Proof: Satisfaction of security Definition 2.5.3

(i) *Curious Authority-resistance*: It is obvious that the verifier is unable to link authentication tags to elements due to the GAAP's *Anonymity* and *Unlinkability* properties;

(ii) *Impostor-resistance*: Due to the collision check in Protocol 2.5.4 step 6, no element can spoof others by fetching the same tags. Furthermore, collusive elements cannot acquire any more tag information other than the ones selected by their own.

(iii) *Eavesdrop-resistance*: The obfuscated tags are encrypted by each element's public key, and so a MITM cannot acquire any information from the channel. ■

Note: It is notable that the proposed GAAP does not resist the verifier and IM collusion. Filling this gap will be the future work of this protocol.

The GAAP Hardware Primitive

We now present the GAAP hardware design details. The hardware primitive implementation as a standalone module enables convenient add-on deployment option in existing connected device network systems. The add-on consists of three components: an initializer module, a plug-in for each component/device, and a plug-in for the verifier.

Initializer Module (IM)

The only task of initializer module (IM) is to generate and distribute random numbers. It does not participate in any other activity. An IM consists of two sub-modules as

shown in Fig. 2·13: a random number generator (RNG), and an encryption unit (ENC) to carry out the public key **Enc()** function.

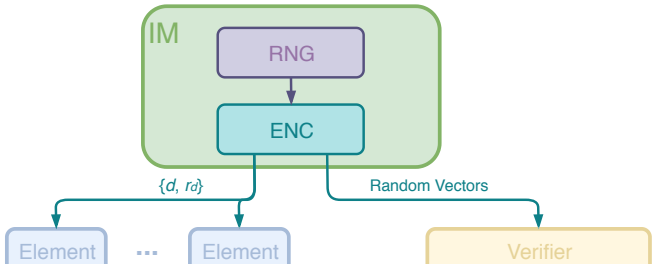


Figure 2·13: The IM functions as described in Protocol 2.5.4 step 4).

Authentication Module for Elements

As shown in Fig. 2·14, the first task of an element's plug-in is to coordinate with other elements for index selection agreement in Protocol 2.5.4 step 3).

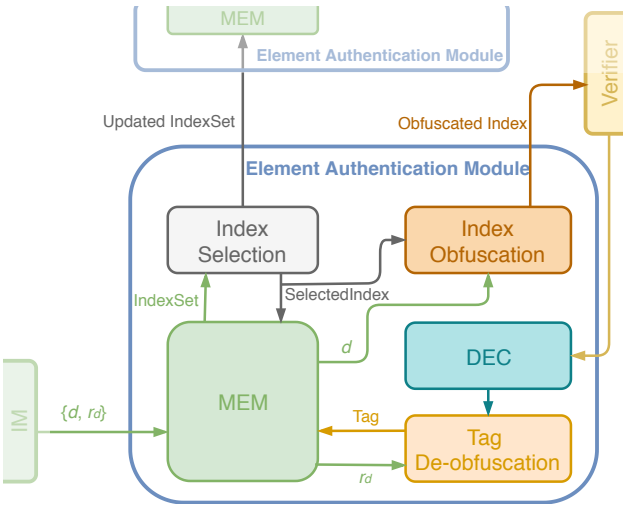


Figure 2·14: The element authentication module.

The Index Selection function is described as Algorithm 4.

Then the SelectedIndex c_j is obfuscated by [Eq. 2.13] in Protocol 2.5.4 step 5) before being sent to the verifier. In Protocol 2.5.4 step 9), on receiving the encrypted

Algorithm 4: Index Selection Agreement Algorithm

```
1 IndexSet = [0 to n-1]
2 for (from Elm_0 to Elm_{n-1}) do
3 |   SelectedIndex = random.choice(IndexSet)
4 |   IndexSet = IndexSet.remove(SelectedIndex)
5 end
```

authentication tags, the DEC and Tag De-obfuscation modules are able to retrieve the targeted tag t_{c_j} by **Dec()** function based on the public-key scheme and de-obfuscation [Eq. 2.14].

Authentication Module for the Verifier

In the verifier authentication module as shown in Fig. 2·15, the Tag Generator is practically another RNG, generating nm authentication tags as in Protocol 2.5.4 steps 1) and 2).

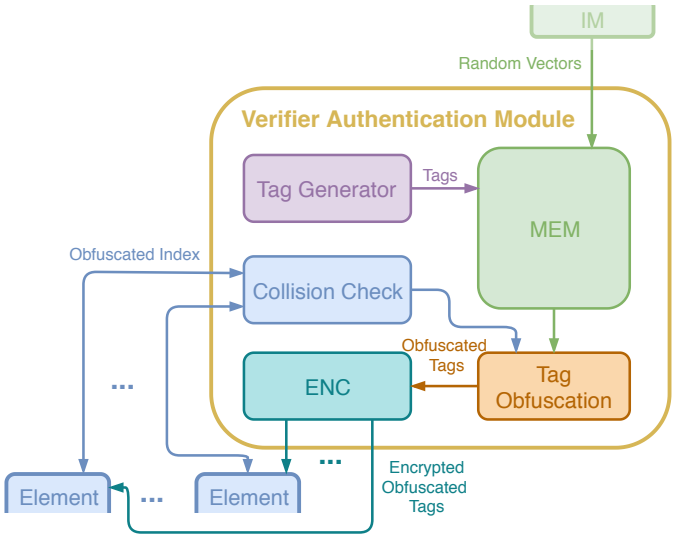


Figure 2·15: The verifier authentication module.

On receiving the obfuscated index selections from devices, the Collision Check functions (described in Algorithm 5) as Protocol 2.5.4 step 6). Since the obfuscated index selection is a bijection to the original index set, we are able to adopt an efficient collision check algorithm. First, each obfuscated index is assigned with an extra

FlagBit. The FlagBit is initialized to 0, indicating this vector has not been visited yet. Then the Collision Check module traverse the obfuscated index set and uses each element as a pointer to visit another. If any element is found to be visited more than once, a collision is detected.

Algorithm 5: Collision Check with time complexity $O(n)$

```
   Initialization:
   ObIndexSet = [n obfuscated indexes]
   for  i = 0 to n do
1  |   ObIndexSet[i].FlagBit = 0
2  end
3  for  i = 0 to n do
4  |   if ObIndexSet[ObIndexSet[i]].FlagBit == 0 then
5  |   |   ObIndexSet[ObIndexSet[i]].FlagBit = 1
6  |   end
7  |   else
8  |   |   Report collision on ObIndexSet[i]
9  |   end
10 end
```

With the previously received n random vectors, the Tag Obfuscation module performs [Eq. 2.13] in Protocol 2.5.4 step 7). [Eq. 2.13] is a critical operation to ensure the delivery of t_{c_j} and concealment of other tags. The obfuscated tags are then encrypted by the ENC module before being sent back to the devices, as in Protocol 2.5.4 step 8).

2.6 Architecture Support

The design methodology behind the proposed architectural model is to provide hardware-supported mechanisms for designer/user-defined security rules and their enforcement in heterogeneous multicore system-on-chip (SoC). It effectively decouples the security or trust level management of processing cores from the integrated SoC.

2.6.1 Hardware Implementation of the Network Interface

The hardware template depicted in Figure 2·16 can be described as the high-level node module of the new architectural model. Each processing node has: (1) a processor socket to support integration of third party processing IPs; (2) a network interface module supporting the novel secure computing protocol; and (3) a virtual channel router. A major contribution of this work is the complete decoupling of the system level fine-grained security management scheme from the processing elements (cores or tenants) and executing software security level. Any processing unit executing any type of software can be installed in the processor socket. Similarly, the proposed security and trust models are oblivious to the on-chip network router microarchitecture.

The hardware modification is constrained to the *Network Interface* (NI) module. The NI is responsible for converting data traffic coming from the local processor and cache subsystem into packets/flits that can be routed inside the network, and for reconstructing packets/flits into data traffic at the opposite side when exiting the NoC. The new network interface has two datapaths: one encrypted and one bypass. The encrypted datapath's functional block description is provided in Section 2.6.2. The *Bypass path* through the NI module enables the disabling or power-gating of the encryption function at a given core site.

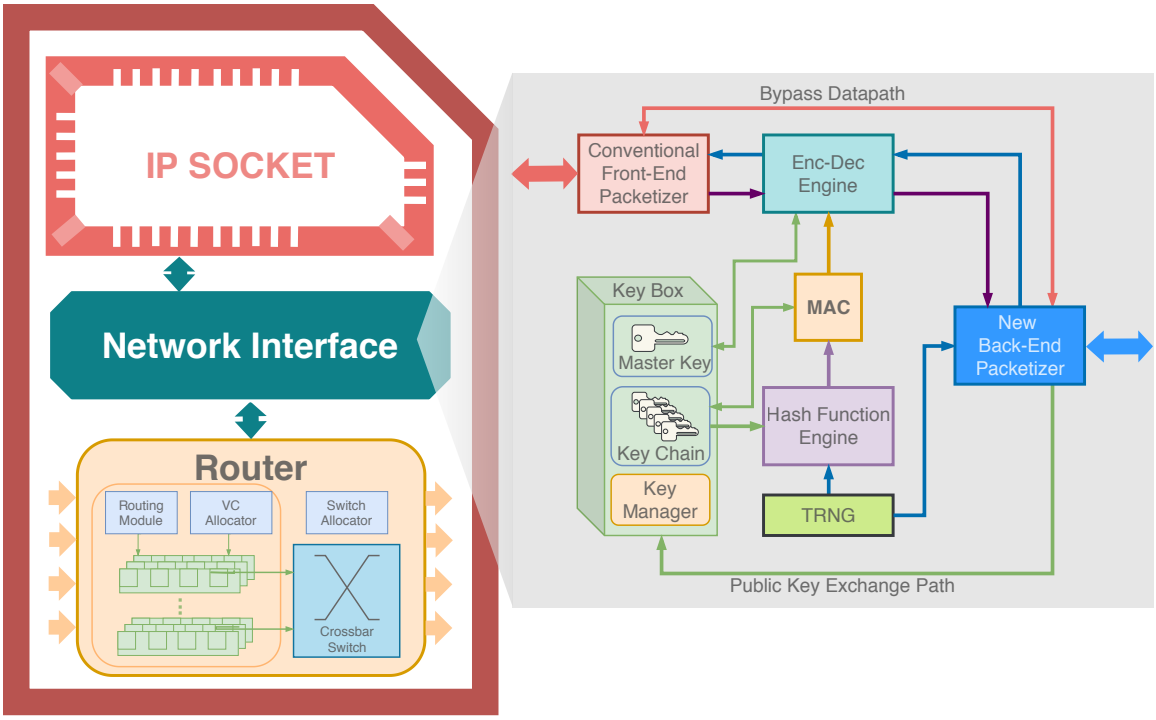


Figure 2·16: The proposed architectural model. A new Network Interface is the key component of it. All of the security features of the system are independent of the processing unit.

2.6.2 Communication Protocol

The communication in the proposed architectural model is an encryption-based scheme. Its communication protocol is as follows:

1. A local processor unit generates message traffic consisting of memory load and store operations, cache coherency messages and inter-core communication traffic.

2. The Front-End Packetizer converts the processor produced data traffic into packets that can be used for communication with the *Network Interface* (NI).

3. The appropriate communication key is selected by the *Key Manager* in the NI. All of the keys are stored in the *Key Box*, which also contains the *Key Manager* function block. There is a single master key per processing site stored in the

Key Box. The set of all the other keys is referred to as the *Key Chain*.

4. The message authentication code (MAC) is generated by feeding the key and one random number into the *Hash Function Engine*. MAC is used as session encryption key, so that even the same core-pair can have multiple distinctive communication sessions.

5. The processor generated packets are fed into the *Encryption Engine* with the MAC to be encrypted. We implement AES key encryption algorithm for the actual encoding of packets, given its low hardware logic cost.

6. The *Encryption Engine* uses the MAC as a key to encrypt the data using the AES algorithm.

7. The encrypted packet and a second random number are re-packetized by the Back-End Packetizer.

8. On the receiving side, packets are first depacketized into encrypted packets and random numbers. The random number is used with the communication key to generate the MAC used to decrypt the packet. The blue directional edges in Figure 2·16 show the return side data-flow. □

2.6.3 Trust-Aware On-Chip Routing Algorithm

The on-chip routing is also aware of the logical security islands and tries to either prohibit or limit the traversal of zones by non-member generated traffic. It uses a similar application-aware deadlock-free oblivious routing approach as introduced in (Kinsy et al., 2009). Algorithm 6 describes the added routing function.

Algorithm 6: Trust-aware on-chip routing algorithm

Objective
Minimize intersections across all routing path sets among islands;
A system with a list of processing elements $P = \{p_1, p_2, ..., p_n\}$;
With the following corresponding list of routers $R = \{r_1, r_2, ..., r_n\}$;
Find a set of routes $S = \{S(R_1), S(R_2), ..., S(R_n)\}$;
Such that $\forall\ p_i \in P,\ S(R_i) = \{r_u, ..., r_v\}$ with $1 \leq u < v \leq n$ while minimizing $\forall(i,j)\ S(R_i) \cap S(R_j)$. ;
The association of a processing element p_i to a router r_j is denoted $p_i \triangleright r_j$ (a p_i runtime trust classification depends on the IP core trust level and the security of the program running on the core. ;
$\forall\ p_i \in P,\ \ S(R_i) = \phi$;
for $i \in [1, n]$ **do**
  for $j \in [1, n]$ **do**
    if $(p_i \triangleright r_j)$ **then**
      $S(R_i) = S(R_i) \cup \{r_j\}$
    end
  end
end
while $(\forall\ p_i \in P,\ |S(R_i)| > 1\ and\ \forall(i,j)\ S(R_i) \cap S(R_j) \neq \phi)$ **do**
  if $(\exists(p_i, p_j) \mid S(R_i) \cap S(R_j) \neq \phi)$ **then**
    if $((|S(R_i)| > 1) \wedge (|S(R_j)| > 1))$ **then**
      $$S(R_i) = \begin{cases} S(R_i) - \{S(R_{min}) \cap S(R_i)\}\ where \\ S(R_i) \subsetneq \{S(R_{min}) \cap S(R_i)\} \\ \{r_e\}\ for\ any\ r_e \in S(R_i)\ otherwise \end{cases}$$
    end
  end
end

2.6.4 Illustrative Example

Figures 2·17 and 2·18 show the case where the data placement for an application task and read and write operations dynamically create a security group. The initial placement is shown in Step 1. The effect of a read-only request from PE 2 is presented in Step 2. A read/write request from PE 4 causes the group to expand (Step 3). In Step 4, an attempt to read/write by PE 6 through PE 2 fails (red edge), since PE 2 cannot be a sponsor. Sponsorship consists of authorizing other threads/tasks/processes to read or write a copy of data without informing the initial owner. Consequently, PE

6 had to join the security group through PE 1 (Step 5). In Step 6, PE 8 is able to join the group through PE 6 and bypass PE 1's sponsorship.

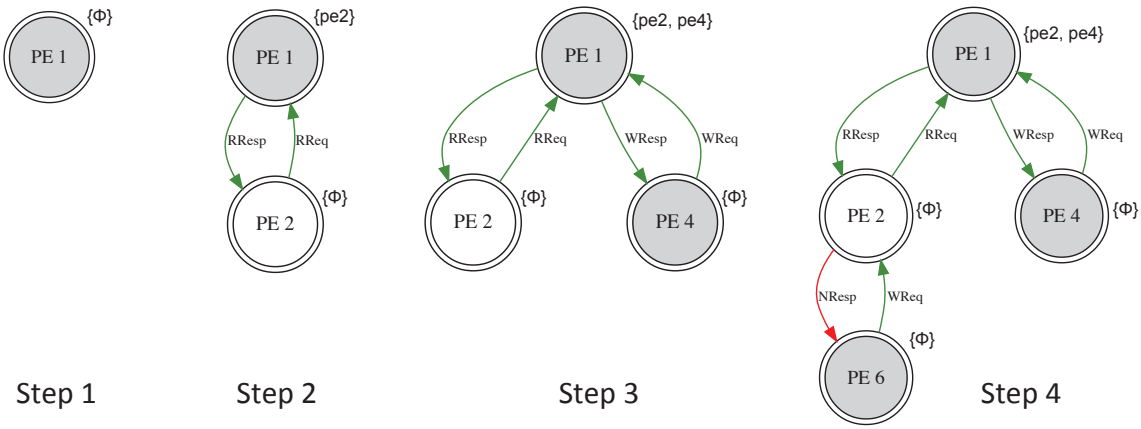


Figure 2·17: Group-forming example (Steps 1–4).

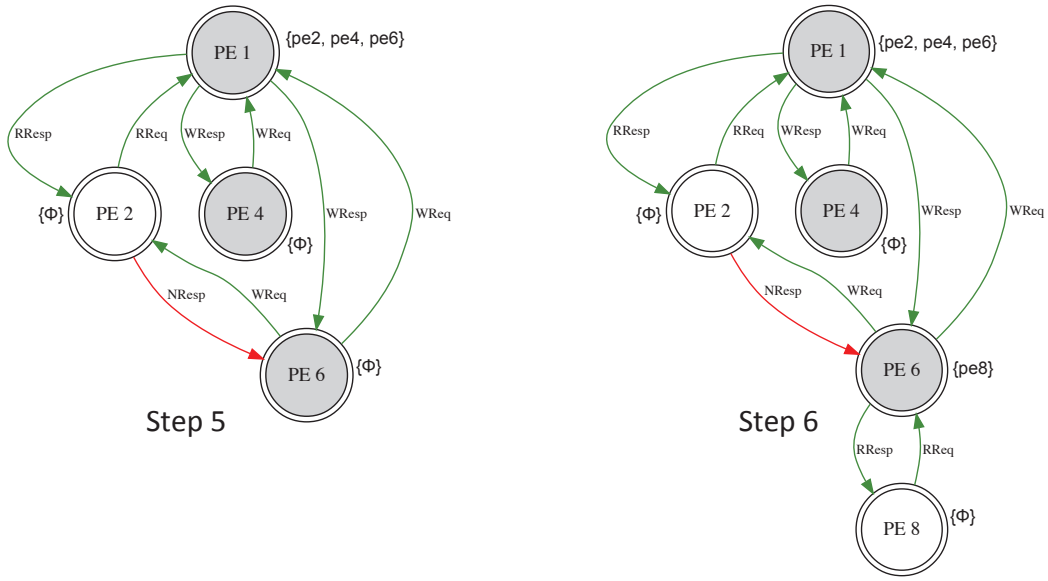


Figure 2·18: Group-forming example (Step 5–6).

As shown in Figure 2·19, PE 1 has no record of the sharing between PE 3, 5, 7 and PE 4. This is done to avoid updating the whole group structure on every entry or exit. Therefore, the control of the security policies becomes distributed.

Changes are more localized and the protocol is more resilient to notification propagation delay associated with group membership alterations. The evaluations show that this distributed approach is more scalable than the full broadcast scheme with equal security guarantees.

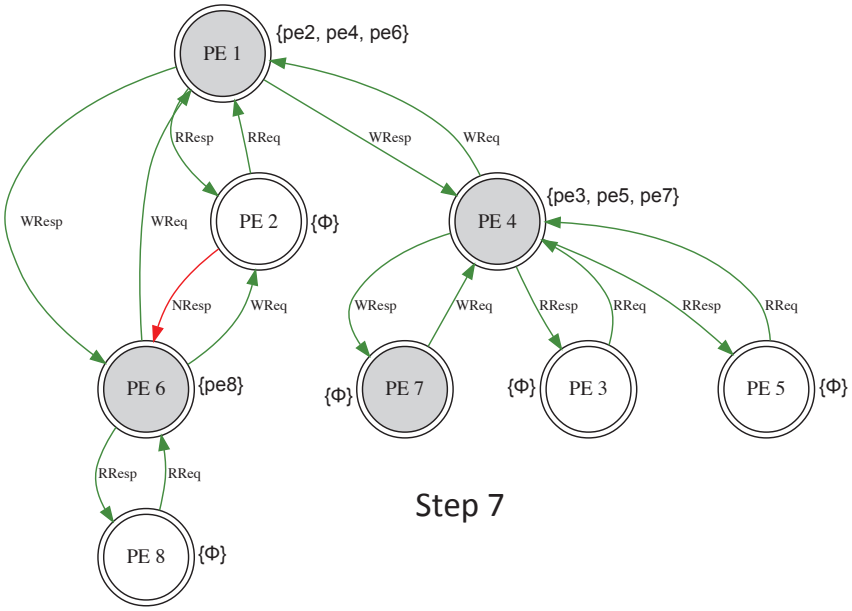


Figure 2·19: Group-forming example (Step 7).

2.7 Conclusion

To solve the problem of mixed security of software and hardware components, in this chapter we develop: (1) a new architectural model which is a generalized multi-tenant, multicore computer architecture with virtual logical zones to enforce isolation by trust levels; (2) a dynamic key management protocol that lends itself well to the architecture for secure efficient heterogeneous computing; and (3) a set of schemes to detect and tolerate dishonest processing elements (PEs) in the island *join* procedure, while supporting privacy preservation for the trusted PEs.

The new architectural model isolates flows and processes based on their trust

levels, effectively creating system-level control access to shared resources: (a) shared memory regions in the network, i.e., virtual channels at the router; and (b) distributed shared main memory modules. In the various sub-parts of the system, the architecture monitors processing traffic to verify their compliance to the trust level security policy in effect. The new architectural model is currently limited to offline classification of programs and processing elements. In addition, the set of rules governing the mapping of programs to cores must be defined beforehand. The architecture has no runtime learning and classification of threats or trust level re-assignments.

Chapter 3

Hardware Root-of-Trust Security Primitives

3.1 Introduction

As part of the SoC security, authenticating and unique identifying of PEs or other hardware components is a critical procedure. An identity of a PE indicates its trust level and so the memory access privilege. In other words, the PE authentication determines the trustworthiness of the initial stage in dynamic *island* forming.

The conventional approach of hardware authentication is usually to attach a static secret key or ID to the device. However, this static approach suffers from several disadvantages. First, it takes an expensive non-volatile memory (NVM) to store such a piece of secret and the secret can be maliciously re-programmed. Second, many identity tags (such as RFIDs, barcodes, and MAC addresses etc.) are separately produced apart from the devices they attach to. Thus a verifier or user is unable to distinguish between a genuine product and a counterfeit with a genuine ID. Finally, under most occasions a device or a person is identified by an ID or a key, which is essentially a string of numbers. A malicious party will be able to spoof the legitimate device or person if it acquires this static string (or even just part of it, such as the last four digits of a credit card number of social security number).

Therefore, a more robust approach for authentication and key storage/transmission is desired. This approach should be inexpensive, naturally integrated in the

device, unpredictable and unclonable, and support dynamic verification. Under this demand the physical unclonable function (PUF) was proposed. A PUF is a piece of hardware that upon given challenges (inputs to the hardware), unique responses will be produced due to its intrinsic manufacturing variation. Each PUF's output (response) is a non-linear function of the outside input (challenge) and the PUF's own physical and unique diversity, in another word, "Silicon Fingerprints" (EE Times, 2010). To apply PUF in the security field, it also needs to be easy to manufacture but hard to duplicate, even under exact the same circuit layout and manufacturing procedures. Because of its attributions of randomness and uniqueness, PUFs can be used for secret keys generation, authentication (i.e. IC, user, product authentication), and identification. In this dissertation we use PUF as the unique identifier and key storage of PEs in SoCs. In distributed systems, we use PUF as the authentication and key agreement primitives among devices (Bu et al., 2018a) (Bu et al., 2018b).

The authentication protocol leveraging PUF's Challenge and Response Pairs (CRP) is shown in Protocol 3.1.1.

Protocol 3.1.1. Denote CHL_i as the i^{th} challenge (input) to a PUF and RSP_i the corresponding response (output). The authentication or identification procedure of PUF is as follows:

i Before a PUF is released, the verifier will challenge it with $\{CHL_i\}$ and store its $\{RSP_i\}$ as a set of CRPs;

ii After a PUF is released and needs to be authenticated, the verifier will send a pre-stored challenge CHL_i to the PUF;

iii When the PUF returns a response RSP_i' to the server, this RSP_i' will be compared with the verifier's pre-stored RSP_i to verify it validity. □

Beside authentication, PUF can also be used to facilitate key agreements (Günlü et al., 2018) (Chatterjee et al., 2018) (Bu et al., 2018c). Protocol 3.1.2 shows a simplified PUF-assisted key agreement.

Protocol 3.1.2. Denote CHL_i as the i^{th} challenge (input) to a PUF and RSP_i the corresponding response (output). The key agreement between Alice and Bob is as follows:

i Before a PUF is released to Bob, Alice will acquire a set of CRPs of the PUF;

ii When a new key needs to be established between Alice and Bob, Alice will select an arbitrary response of the PUF as the new key, e.g., RSP_i. Alice then sends the related challenge CHL_i to Bob;

iii Bob uses this CHL_i to challenge his PUF and generates RSP_i, which is now the commonly shared secret key between Alice and Bob. □

Although this concept of physical uniqueness has been known since 1983 (Bauder, 1983), the term PUF only came to be in 2002 (Gassend et al., 2002b). And from 2010, PUF has become more popular and drawn large attention. Since then, researchers have been studying and developing different types of PUFs and their implementations.

PUFs can be categorized in several divisions. Based on how the randomness is introduced, there are PUFs using explicitly-introduced randomness, and PUFs using intrinsic randomness.

3.1.1 PUFs Using Explicitly-introduced Randomness

For this type of PUFs, the manufacturing variation is directly controlled by artificially introduced randomness. Thus devices tend to have more distinguishing behaviors. Optical and coating interventions are mostly used in this type (Herder et al., 2014).

- Optical PUF: It uses a laser beam to shine onto a material doped with light scattering particles, which brings random and unique speckle patterns;
- Coating PUF: It is built upon the top layer of an IC with randomly doped dielectric particles. Its randomness comes from the capacitance between such metal wires.

3.1.2 PUFs Using Intrinsic Randomness

PUFs using intrinsic randomness are highly attractive because they can be included in a design without modifications to the manufacturing process. Moreover, this property makes PUF also manufacturer-resistant. Meaning even manufacturers cannot manipulate its manufacturing procedure to produce two identical PUFs.

For this reason we will focus more on this type in the dissertation. Based on the source of randomness, there are several types of PUFs in this category:

- Delay PUF: It uses the random variations in delays of the basic elements of the circuit (gates, LUTs, etc.). Essentially, the uniqueness comes from the slight size variation of each transistors. Given an input (challenge), a race condition will appear in the circuit, and different chips with the same circuit will have different delay patterns. Some implementations of this kind are ring oscillator-based (RO) PUF, bistable ring PUF, and multiplexer and latch-based Arbiter PUF;

- Memory PUF: Almost all ICs have memories on them and they can be used for IC authentication or identification. The output of a memory PUF is the boot up state (1 or 0) of each memory cell, which varies from device to device. A bit enrollment algorithm is usually involved to select the "stable" memory cells (the cells having consistent boot up states every time) for the PUF output. Both SRAM and DRAM have been proposed to be used as PUFs. Also butterfly PUF based on cross-coupling of two latches has already been commercialized;

- Mixed-signal (Analog) PUF: It quantizes an analog signal (drain currents, threshold voltage etc.) to produce a digital response.

Besides these common implementation of PUFs using intrinsic randomness, there

are also other types: magnetic PUF made from magnetic stripe card, metal Resistance PUF, and quantum confinement PUF etc.

3.1.3 Strong and Weak PUFs

Based on the size of the challenge-response pairs, PUFs can be categorized as weak and strong PUFs, which have different applications in security.

Weak PUF

The challenge-response pair (CRP) set size grows linearly with the PUF size. A weak PUF usually only has one CRP (such as the memory PUF), and so its CRP should be secured and not accessible by untrusted parties;

Strong PUF

The CRP set size grows exponentially with the PUF size, mostly used for dynamic authentication. For strong PUFs, even a large set of CRP is made publicly accessible, it should still be extremely difficult for an adversary to predict any unknown CRPs.

3.1.4 PUF Modeling and Secure PUFs

Although PUFs are supposed to be unclonable and non-reproducible, researchers have proposed ways to clone PUFs and predict their responses. One popular approach is to use statistical tools such as machine learning or algebraic techniques (Rührmair et al., 2013).

Several secure constructions were proposed to add more non-linearity to PUF designs in order to resist modeling attacks. Lightweight secure PUF (Majzoobi et al., 2008) was among the first of the proposals. In a secure PUF, several Arbiter PUFs are placed together to create one response bit. Each of the Arbiter PUFs receives the obfuscated challenges by a strict avalanche criterion (SAC) network and the responses

of these PUFs are then XORed and combined with transformed challenges to build the final response bits.

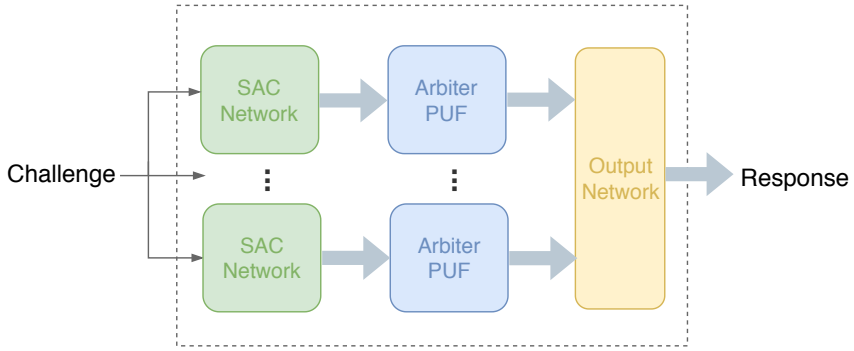


Figure 3·1: The circuit above produces 1 bit of PUF response.

However, later research efforts show that when modeling attacks are assisted by side channel attacks on the PUF's helper data (Delvaux and Verbauwhede, 2013), most PUF's including lightweight secure PUFs can still be modeled.

Therefore, in addition to secure hardware designs, researchers have also proposed algorithmic approaches to strengthen PUF's unclonability. Hashing the challenges or responses of a PUF (Gassend et al., 2002a) was shown to be effective since machine learning so far has not been able to learn cryptographic hash functions. Adversarial machine learning was also adopted as a cost-efficient approach to secure PUFs against modeling attacks (Nguyen et al., 2018) (Yu et al., 2016) (Yu et al., 2011).

3.1.5 PUF Implementation Challenges

It is notable that the same challenge should receive different responses from different PUFs, even though those PUFs are built from identical design on identical circuits. Also ideally, different responses should have as large a Hamming distances as possible. Usually non-cryptographic hash functions are used to randomize the responses to amplify their variations.

Also due to some unstable factors (temperature, voltage etc.), there is usually

noise in PUF's response. Therefore error control coding and helper data are used to remove the noise.

Despite the many existing research efforts on PUF's high level architectures, PUF's low level implementation is still non-trivial. Without a proper design and configuration, regardless how elegant the high level architecture is, the PUFs will lose their uniqueness. Namely all the PUFs built from the same circuit will return the same responses upon a challenge. Although much research has been done in this area on Xilinx ISE, there have been few for the new IDE Vivado due to its inconvenient FPGA editing. Therefore, this chapter will discuss these problems and provide solutions and improvements to these PUF implementations on FPGA. The major contributions of this chapter are:

1. It explores the implementation problems and solutions in the new Vivado IDE;

2. It suggests several improvements to the existing delay PUF designs to enhance reliability;

3. It proposes a novel type of PUF named multi-identity PUF (Mi-PUF). Unlike ordinary PUFs which attest only one legal identity of an element, the Mi-PUF is able to demonstrate multiple legitimate identities from one element. Mi-PUFs can be used to preserve the privacy of an element while proving its legitimacy.

The rest of the chapter is organized as follows. Sections 3.2 gives a brief introduction of several delay-based PUFs' architecture. Section 3.3 explores the critical problems of their implementations on FPGA with the latest IDE. Section 3.4 proposes an improved design for the RO PUF. Section 3.5 explores the routing and placement of RO and Arbiter PUFs in the latest IDE. Section 3.6 proposes the Mi-PUF based on both RO and Arbiter PUFs. And finally Section 3.7 evaluates the Mi-PUF's quality.

3.2 The Architectures of Delay-based PUFs

In this section several popular designs of delay-based PUFs are introduced, based on which Section 3.3 will dig deeper into their implementations. For brevity we refer to delay-based PUFs as delay PUFs from now on.

3.2.1 Delay PUFs

We focus more the delay PUFs because they generate much larger CRP sets than memory PUFs which are vulnerable under replay or probing attacks. With delay PUFs there can be a new challenge for every round of authentication or identification, so that replay attacks will never apply. Delay PUFs can be used both for authentication or secret key generation. In addition, most secure PUFs are also based on the basic delay PUFs. To better facilitate the presentation, we introduce the following notations:

- CHL: a PUF's challenge;
- RSP: a PUF's response;
- c: the number of bits in a PUF's CHL;
- r: the number of bits in a PUF's RSP;
- n: the number of ROs in a RO group which produces 1 bit of response;
- $|CRP|$: the size of a PUF's CRP set;

The commonly seen delay PUFs are: Ring Oscillator (RO) PUF (Suh and Devadas, 2007) (Maiti et al., 2012), Arbiter PUF (Jae W et al., 2004) (Lim et al., 2005), Glitch PUF (Shimizu et al., 2012) (Anderson, 2010), HELP (Aarestad et al., 2013) and etc. Among them the most popular ones are the RO and Arbiter PUFs. The challenges are the inputs of the circuit, and the responses are generated by the delay differences of the circuit.

Ring Oscillator (RO) PUF

A RO PUF consists of multiple inverter chains, and in each chain odd number of inverters are connected in a loop. Due to the manufacturing variation resulting in each inverter's delay, each oscillator has a different and unpredicted frequency from others. At the end of this circuit there are two counters of the same size whose increment speed depends on the frequencies of the corresponding chains, which serve as the clock. A 1-bit RO PUF is shown in the Fig. 3·2. The challenge will be the input to the decoder selecting two ROs from the group of n ROs, and the response is the comparison of the values of the two counters clocked by the selected ROs.

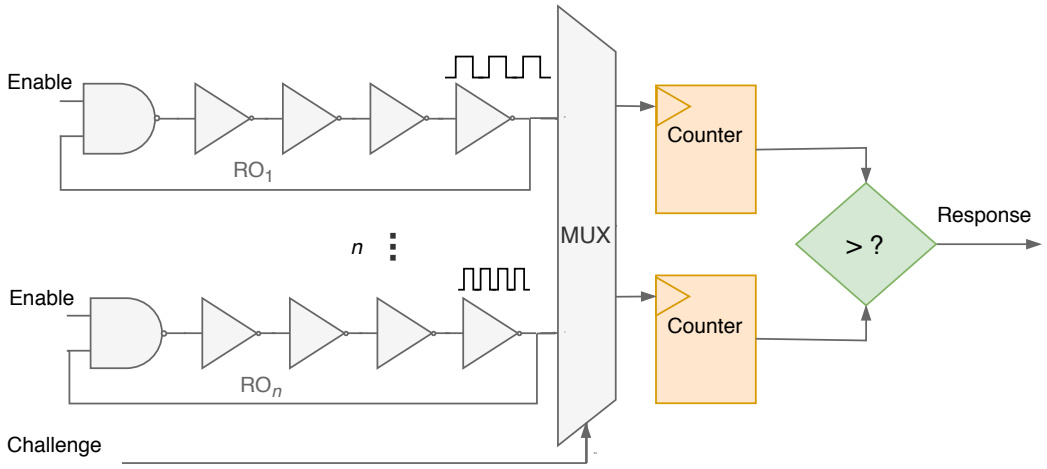


Figure 3·2: The circuit above produces 1 bit of RO PUF response. To generate more response bits, more RO groups and counters can be added.

Suppose for a RO PUF, there are r RO groups and in each group there are n ROs. If the first and second half of the challenge CHL is used to select two different ROs in this group, then for the challenge size c is:

$$c = 2 \cdot \lceil log_2 n \rceil. \tag{3.1}$$

The total number of CRP is:

$$|CRP| = \binom{n}{2}. \tag{3.2}$$

While the Arbiter PUF requires highly symmetric implementation, the RO PUF does not, which makes it easier to be implemented on FPGAs, although it still requires identical routing and relative placement for each RO. In the FPGA implementation, each inverter is instantiated by a 1-to-1 lookup-table (LUT), and the NAND gate by a 2-to-1 LUT.

Arbiter PUF

Arbiter PUF is multiplexer (MUX) and D-flip-flop (DFF)-based. It takes two sets of parallelly connected MUXes and the output is latched by a DFF as shown in Fig. 3·3. As a positive edged signal is applied to the first two MUXes, the binary challenge vector determines which two paths the signal will go through. Because of the unique intrinsic delay in each MUX, the positive edged signal will create a racing condition between the two paths to the clock port and data port of the DFF. The output of the DFF is then a random response bit as shown in Fig. 3·3. In this case the two parallel MUX chains' wiring should be completely symmetric, so that the only factor that determines the response will be the MUXes' manufacturing variations.

For an Arbiter PUF, if there are r MUX chain pairs, and the challenge has c bits, then the total number of CRP is:

$$|CRP| = 2^c. \tag{3.3}$$

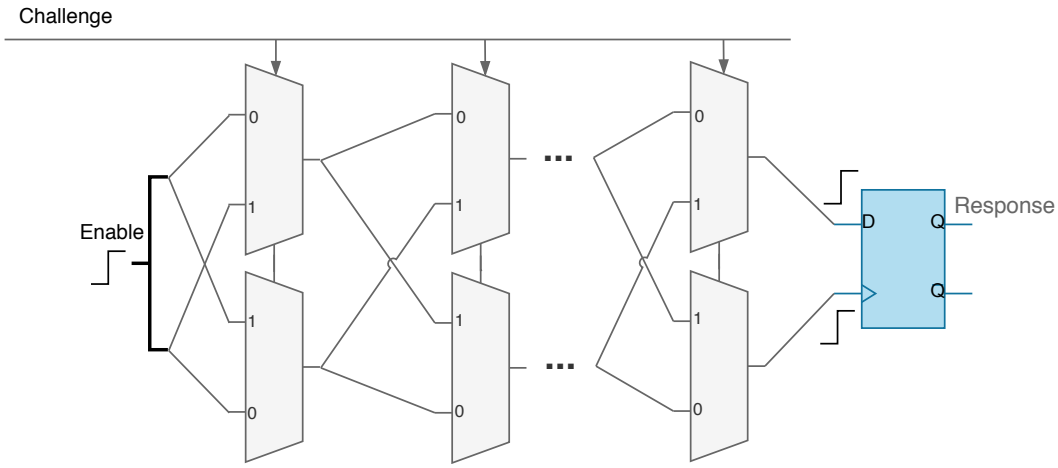


Figure 3·3: The challenge vector determines whether the rising signal will reach the DFF data port or the clock port. In the former case 1 will be the response, and 0 will be the latter case. The circuit above produces 1 bit of PUF response. To generate more response bits with the same challenge input, more MUX chains can be added.

3.3 Implementation Problems and Solutions

Previously, most researchers prefer to use the combination of Xilinx ISE + Xilinx FPGA because ISE allows users to configure the placement and route of the circuits using its FPGA Editor (Soybali et al., 2011) (Morozov et al., 2010) (Khoshroo, 2013). However, from October 2013 ISE is no longer updated and most researchers in the FPGA field have moved to the Vivado IDE (latest version 2017.2). Nevertheless, since then there are few works being done with clear presentation of PUF design methodology with this new IDE. One of the major reasons is that Vivado does not have the convenient FPGA Editor anymore and users need to find the alternatives such as XDC macro or hard macro.

Since the new Vivado IDE creates many unique problems comparing to its predecessor, in this chapter we will illustrate the problems and solutions of PUF design in this new environment. We will also give detailed instructions and improvements on PUF's design and implementation, in order to assist the researchers who need to

work with new FPGA models and IDE without the FPGA editor.

We will focus on the design and implementation of delay PUFs, especially the basic RO and Arbiter PUFs. Once these fundamental elements are figured out, one can make modifications easily based on them to produce other delay PUFs.

The following three sections are organized as follows. Sections 3.3.1 and 3.3.2 are about the common errors designers could encounter in Vivado. Section 3.4 is on the timing issue of the comparison between two RO's counters. Section 3.5 explores the fixed relative routing and placement in the new Vivado IDE.

3.3.1 Handling Design Rule Violations

Both Arbiter and RO PUFs involve violations against certain design rules. These violations need to be properly allowed otherwise Vivado will refuse to proceed with synthesize or implementation.

Combinatorial Loops

Taking a closer look at a single ring oscillator (RO), it consists of odd number of inverters as in Fig. 3·4:

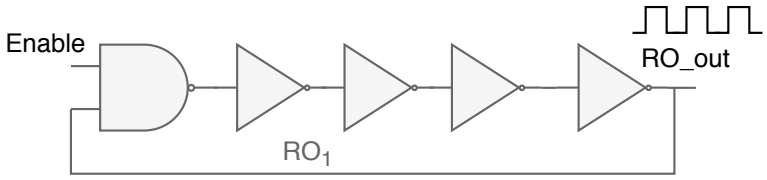


Figure 3·4: A ring oscillator consisting of 5 inverters (NOT gates).

When "Enable" is set to 1, this inverter chain will produce a $1\to0\to1\to0\to1\cdots$ sequence similar to a clock signal. However, since the RO_out signal is fed back to the NAND's input pin, it forms a combinatorial loop which is not permitted by the synthesis tools.

To allow a combinatorial loop, the following line should be added before the instantiation of a RO module:

```
(* ALLOW_COMBINATORIAL_LOOPS = "TRUE" *)
```

To further secure the green light to allow combinatorial loops in the *.bit* file, firstly a *.tcl* file should be created with the following tcl commands:

```
set_property SEVERITY {Warning} [get_drc_checks LUTLP-1]
```

This will downgrade the combinatorial loop error to an ordinary warning so that the bitstream can be written.

Then in the Vivado IDE's Project settings → Bitstream → tcl.pre*, the *.tcl* file needs to be loaded prior to device programming in order to execute this severity downgrade.

Gated-clock

In the RO PUF, since the counter is clocked by a combinational logic, it forms a so-called "gated-clock" as shown in Fig. 3·5. Gated clocks can cause glitches, increased clock delay, and clock skew. Therefore designs of this type are not encouraged by the synthesis tools. In some cases, it can cause a critical error preventing the synthesis or implementation process.

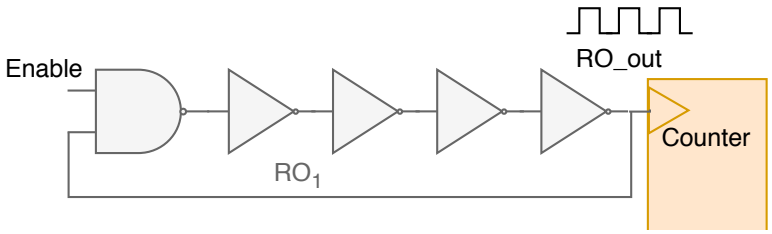


Figure 3·5: The counter is driven by the combinational logic of RO.

Similar problem also exists in the Arbiter PUF. In Fig. 3·3 one of the signals drives the clock port of the arbiter (DFF). Gated-clock can cause errors in the synthesis,

implementation, and bit stream writing processes. In order to accomplish these processes, the severity of this error also needs to be downgraded as the combinatorial loop error.

To do so, in the *.tcl* file mentioned previously, the following tcl command should also added:

```
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```

This command will set the gated-clock error down to a ordinary warning so that the bit stream can be written.

3.3.2 Preventing Logic Trim

As shown in Fig. 3·4 that a RO consists of odd number of inverters. However, this circuit is logically equivalent to a single inverter when Enable is set, which will be the result of synthesis' optimization as in Fig. 3·6.

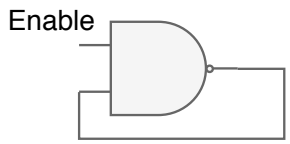


Figure 3·6: A 5-stage RO is logically trimmed to a single NAND gate.

To preserve the inverters in RO from being optimized away, the attribute of "KEEP" is commonly suggested. However, the "KEEP" attribute is not able to preserve elements from logic trim in post-optimization of the synthesized design.

In order to preserve the entire inverter ring, the following attribute is needed before the declaration of each inverter and wire, as well as the RO module instantiation (Xilinx, 2018b):

```
(* DONT_TOUCH = "TRUE" *)
```

Now for the RO module instantiation, before it there should be two attributes:

"DONT_TOUCH" and "ALLOW_COMBINATORIAL_LOOPS".

Then in both RTL and technology schematics, every component of the ring is kept:

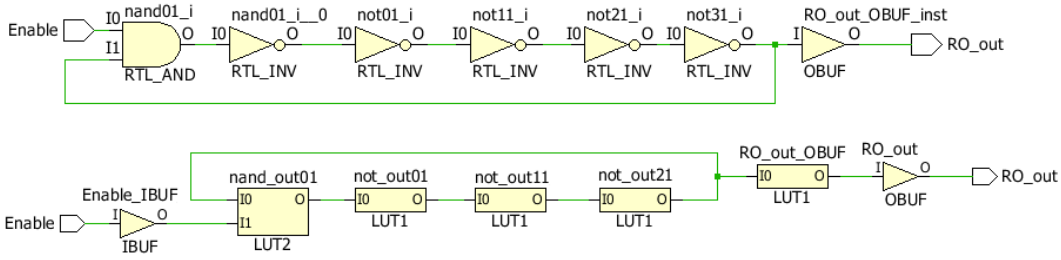


Figure 3·7: The RTL schematics (upper) of a 5-stage RO generated by Vivado synthesis, and the technology schematics (lower) of the same RO instantiated by one LUT2 as NAND, and four LUT1s, as inverters.

3.4 RO PUF: Design of the Stopwatch

There is a moment that the two counters in Fig. 3·2 will be compared against each other at a certain point. However, it cannot be any moment. The comparison needs to be made before overflow occurs to any of the two counters. Otherwise a counter's value at the comparison point might not represent its RO frequency correctly.

Thus there needs to be a stopwatch to fire the comparison signal at the right timing. The stopwatch's period needs to be smaller than both the two counter's overflow periods as shown in the figure below.

However, if the stopwatch's period is larger than any of the counters', the comparison of counter values may not correctly reflect the two RO's speed difference as shown in Fig. 3·9.

Usually the stopwatch module is controlled by a real clock on FPGA. Then its period needs to be set under the knowledge of the RO frequency (counter overflow period). On the other hand, the stopwatch's period also needs to be large enough to

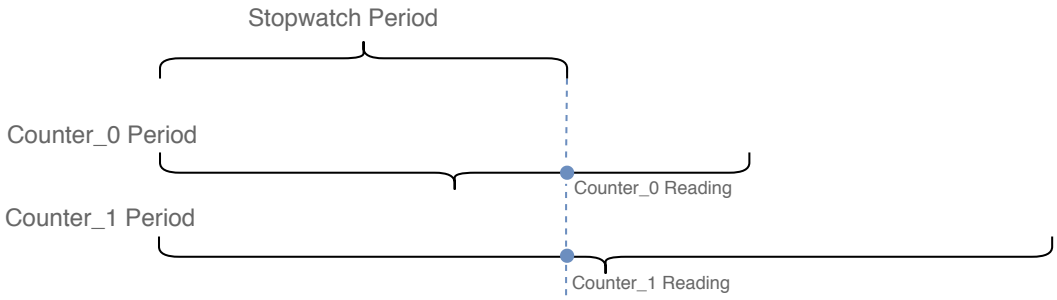


Figure 3·8: If Counter_0's driving RO is faster than Counter_1's, then its overflow period is smaller. When the stopwatch fires the comparison signal, Counter_0 will have a larger reading than Counter_1. Thus a 1 bit response can be generated by comparing the two readings.

reveal the ROs' frequency difference.

3.4.1 The Comparison Timing with the Synchronous Stopwatch

Denote the frequency of the 1^{st} RO as f_{RO_0}, the 2^{nd} RO as f_{RO_1}, the FPGA's clock as f_{FPGA}, and the size of the counters as b-bits. Assume the stopwatch fires the comparison signal at the k^{th} clock cycle, to have the stopwatch module function properly as shown in Fig. 3·8, the following statement needs to be true:

$$\left(\frac{1}{f_{RO\_0}} \ll \frac{k}{f_{FPGA}} < \frac{2^b-1}{f_{RO\_0}}\right) \&\& \left(\frac{1}{f_{RO\_1}} \ll \frac{k}{f_{FPGA}} < \frac{2^b-1}{f_{RO\_1}}\right). \tag{3.4}$$

Note: f_{FPGA} can be acquired from the FPGA manual.

The counter size b can be set as a large value first, say $28 \sim 32$, in order to acquire a measurable overflow period. Then the most significant bit (MSB) of the counter can be set as an output pin which connects to an oscilloscope. By observing the waveform the overflow period can be calculated, as well as the RO frequency. If the overflow period of a counter is denoted as T_{OF}, then:

$$f_{RO} = \frac{2^b-1}{T_{OF}}. \tag{3.5}$$

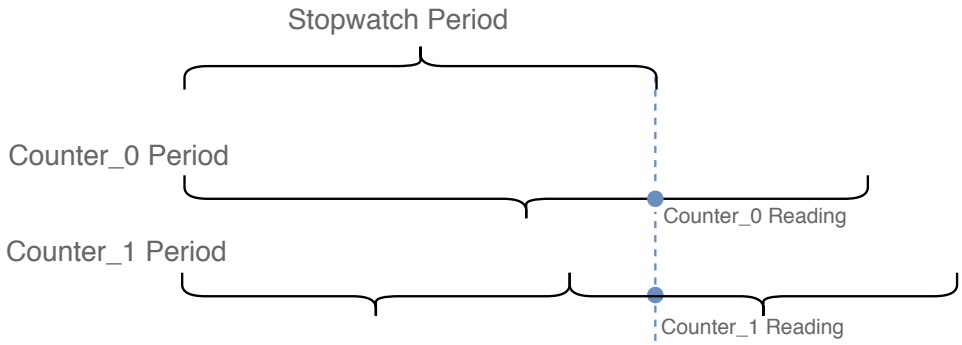


Figure 3·9: If Counter_1's RO is faster than Counter_0's, then it has a smaller overflow period. However, if the stopwatch's period is larger than Counter_1's, then before the comparison signal is set, Counter_1 has already overflown. Then when the comparison happens Counter_1 may have a smaller reading than Counter_0, mistakenly indicating that Counter_0's RO is faster.

With these parameters a proper k can then be selected.

Example 3.4.1. We show an example on how to use measured RO frequencies which can be used to set the comparison timing. The waveform of the MSB (the 28th bit) of RO_0's counter implemented on a BASYS 3 FPGA board is shown in Fig. 3·10 together with another RO_1's counter's. The two ROs are identically routed and placed.

By [Eq. 3.5] the frequencies of RO_0 and RO_1 are:

$$f_{RO\_0} = \frac{2^{28} - 1}{0.7764} = 345,743,759Hz.$$

$$f_{RO\_1} = \frac{2^{28} - 1}{0.7825} = 343,048,505Hz.$$

Therefore RO_0 is faster than RO_1 for about 2.7 MHz and this difference will be demonstrated by the comparison of the counters. □

3.4.2 The Asynchronous Stopwatch

A better approach is to use an asynchronous stopwatch. A MAX value can be preset where $MAX \leq 2^b - 1$. Then whichever counter reaches to the MAX first, both the counters will stop incrementing and then the comparison is made. The advantage of

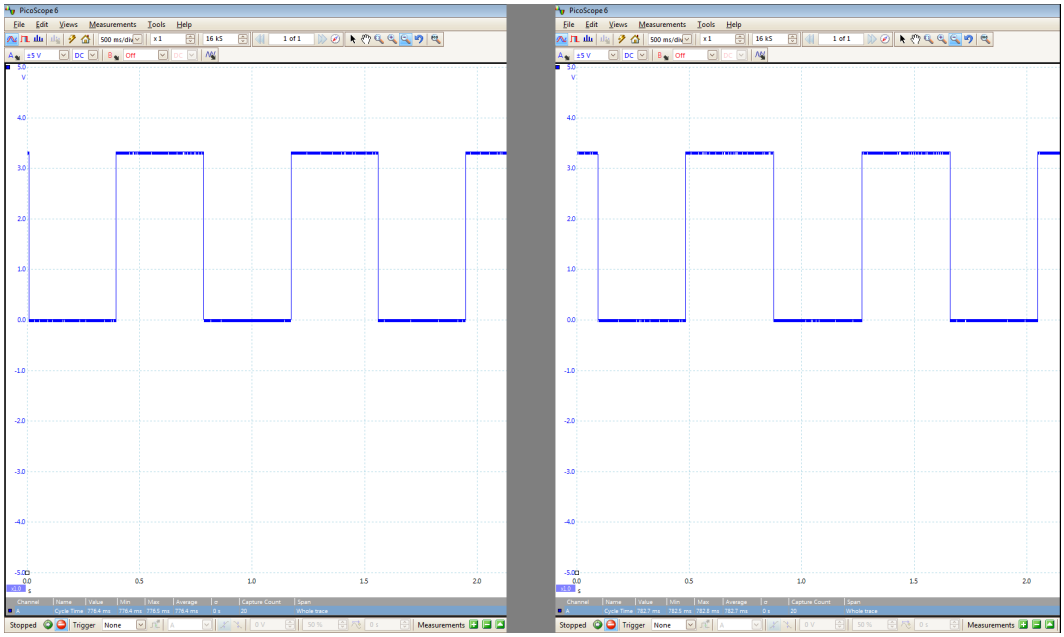


Figure 3·10: RO_0's counter's period is 776.4 ms (left), and RO_1's counter's period is 782.5 ms (right).

this approach is the avoidance of using the FPGA's clock, as well as the complication of measuring and calculating [Eq. 3.4, 3.5]. The only criterion is that b should be a large enough number to reveal the RO's frequency difference. Then the comparison timing graph will be:

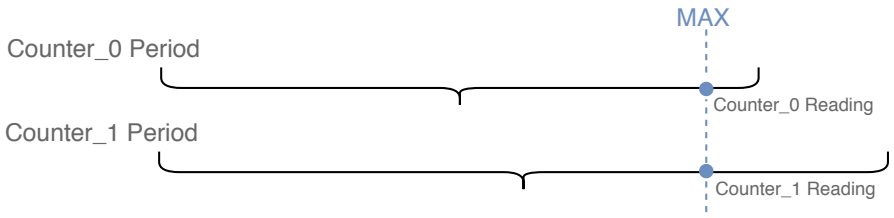


Figure 3·11: If Counter_0 firstly reaches to MAX, then both counters stop and the comparison is made. Vice versa for Counter_1's case.

With the above improvement, overflow of the counters will never be an issue as in Fig. 3·9. In addition, it saves the power of using the system clock and the hassle of identically routing it to Counter_0 and Counter_1.

3.5 Fixed Placement, Pin, and Relative Routing for PUF Implementation in Vivado

A properly functional delay PUF design should generate different responses on different FPGA boards. However, if the routing and placement are automatically carried out by the design tools, then the circuit delay produced by automated routing will dominate over the intrinsic delay of the LUTs. This will result in having the same CRPs on different FPGA boards upon a challenge, namely the PUFs lose their uniqueness.

For example, to produce one bit of response, a RO PUF will make comparison between the frequencies of two identical ROs. Although the two ROs can be instantiated from the same HDL module, after implementation they could be automatically placed and routed in different ways as shown below.

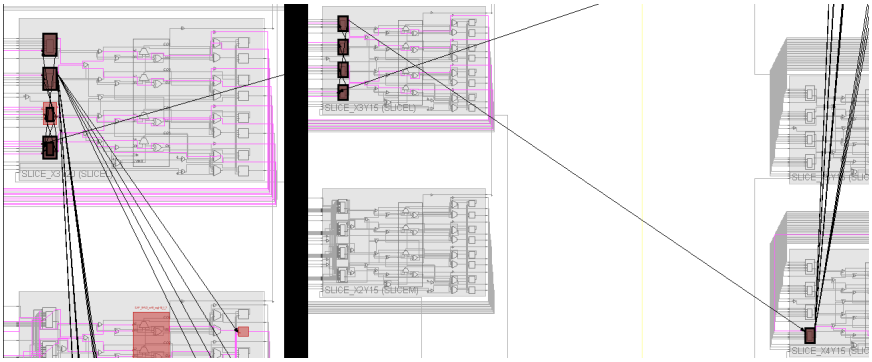


Figure 3·12: RO_0 on the left occupies only one slice, while the RO_1 on the right takes two slices and they are far away from each other. Obviously when a challenge demands the comparison between RO_0 and RO_1, the former will be faster and result in a larger reading its counter than the latter in all FPGA boards.

Therefore the frequencies of the two ROs in Fig. 3·12 are determined essentially by their routing and placement which are identical on every FPGA, rather than the LUTs' intrinsic manufacturing differences, which are unique on every FPGA. Due to

this reason if the generated *.bit* file is programmed to different FPGA boards, although the responses may seem random, all the FPGAs will all have identical CRPs and so the PUFs fail to have their unique silicon fingerprints.

Similar or even more severe situation also exists in the Arbiter PUFs. When the signal of a positive edge is propagated from one stage of two MUXes to the next in two different routes, these two routes have to be symmetrically routed, otherwise the routing delay will overcome the MUXes' intrinsic delay. Moreover, at the end of the stages the two routes will enter into the clock port and the data port of the arbiter. The symmetry of these two paths is even more critical in producing the final response.

To eliminate the timing difference caused by automatic placement and route, all the ROs or the MUXes need to be manually placed and routed identically, so that the LUTs' intrinsic delays will be the only factor affecting the response.

There are three parameters to fix:

1. Placement

   - RO PUF: the 5 LUTs of each RO need to have the same relative placement;
   - Arbiter PUF: the two MUXes of each stage need to have the same relative placement;

2. Pins

   - RO PUF: since different pins of a LUT have different speed, the pins for each LUT in each RO should be locked identically;
   - Arbiter PUF: the propagation of the signals among the stages is shown as Fig. 3·3. For the two MUXes in each stage, the signal will either be propagated from the previous stage to either both of the inputs 0, or both inputs 1 of the current stage. Therefore the inputs 0 of all MUXes should be locked to the same pin, and similarly input 1 to another pin;

3. Route

   - RO PUF: the same relative routing constraints need to be applied to all the ROs and their LUTs within;
   - Arbiter PUF: as shown in the Fig. 3·3, the routing from both outputs of the previous stage to both inputs 0 of the next stage should be the same, as well as the routing from the outputs of the previous stage to both inputs 1 of the next stage.

In Vivado, although the FPGA Editor is no more, users can use the *.xdc* file to specify placement and route. They can also use XDC macros to duplicate the routing of one manually routed RO or MUX to others.

Fixing the Placement and Pins

We will firstly take one RO shown in Fig. 3·4 as an example of the fixed place and route.

Firstly, each RO's NAND gate and 4 NOT gates are all instantiated by the LUTs and referred to as cells in the implementation process. A NAND gate is instantiated by LUT2 which is a 2-to-1 MUX, and a NOT gate by LUT1 which is a 1-to-1 MUX. These LUTs should all have identical relative placement. Knowing that each Xilinx FPGA's SLICEL (logic slice) has 4 LUT6s and 8 flip-flops, the 4 NOT gates for example, can be placed on those 4 LUT6s and the NAND gate on another SLICEL's LUT6. Their relative locations also need to be fixed for every RO. One example is from NAND0 to NOT2 on basic elements (BEL) A5LUT to D5LUT, and NOT3 on the neighboring SLICEL's A5LUT.

The relative placement can be set through the set_property LOC and BEL keyword in the *.xdc* constraints file as (Xilinx, 2016):

```
# Placing each cell into a fixed SLICEL
```

```
set_property LOC SLICE_X0Y0 [get_cells RO_0/NAND0]
set_property LOC SLICE_X0Y0 [get_cells RO_0/NOT0]
set_property LOC SLICE_X0Y0 [get_cells RO_0/NOT1]
set_property LOC SLICE_X0Y0 [get_cells RO_0/NOT2]
set_property LOC SLICE_X1Y0 [get_cells RO_0/NOT3]

# Placing each cell into a relative LUT in that SLICEL
set_property BEL A5LUT [get_cells RO_0/NAND0]
set_property BEL B5LUT [get_cells RO_0/NOT0]
set_property BEL C5LUT [get_cells RO_0/NOT1]
set_property BEL D5LUT [get_cells RO_0/NOT2]
set_property BEL A5LUT [get_cells RO_0/NOT3]
```

Secondly, for a certain cell in all the ROs, its pins should be locked identically. According to the Vivado constraints manual, A6 and A5 are the faster pins than A1 $\sim$ A4. However, as long as each of the five LUTs in different ROs is locked with the same pins, it matters not that which LUT has which pins.

The pins can also be set through the set_property LOCK_PINS keyword in the *.xdc* constraints file as. One example would be:

```
# Lock the input pins
set_property LOCK_PINS {I0:A1 I1:A3} [get_cells RO_0/NAND0]
set_property LOCK_PINS {I0:A5} [get_cells RO_0/NOT0]
set_property LOCK_PINS {I0:A6} [get_cells RO_0/NOT1]
set_property LOCK_PINS {I0:A4} [get_cells RO_0/NOT2]
set_property LOCK_PINS {I0:A2} [get_cells RO_0/NOT3]
```

Fig. 3·13 shows two ROs with the same relative placement and pin locking. Each RO takes up a Complex Logic Block (CLB, consisting of two SLICELs).

Fixing the Routes

Finally, each cell's route needs to be fixed identically through the set_property FIXED_ROUTE command in the *.xdc* constraints file. Since RO PUF does not require symmetric design, as long as any two ROs' routings are the same, they are considered identical.

Firstly, one can either manually route a wire, or let Vivado Implementation do it automatically. The user can leverage the following tcl command to acquire the route

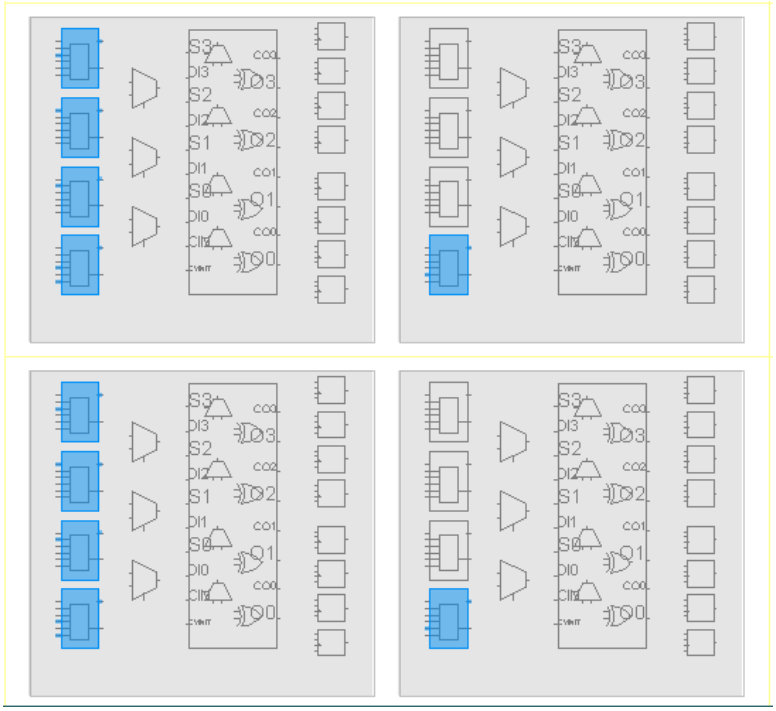


Figure 3·13: Two ROs with identical placements (LUTs highlighted in blue) and pins (highlighted in blue in each LUT).

of that wire. For example, to request the routing path of wire W1 in RO_0:

```
get_property ROUTE [get_nets RO_0/W1]
```

And then the following command can be used to duplicate the same routing to wire W1 in RO_1:

```
set_property FIXED_ROUTE {RO_0/W1's route} [get_nets RO_1/W1]
```

Similarly the other cell's routing can be fixed as well. Fig. 3·14 shows two identically routed ROs.

Although pins, BELs, and relative routing can be duplicated easily to other ROs by set_property, it should be noted that the absolute location property SLICE should be different for each RO:

```
set_property LOC SLICE_XxYy [get_cells ...]
```

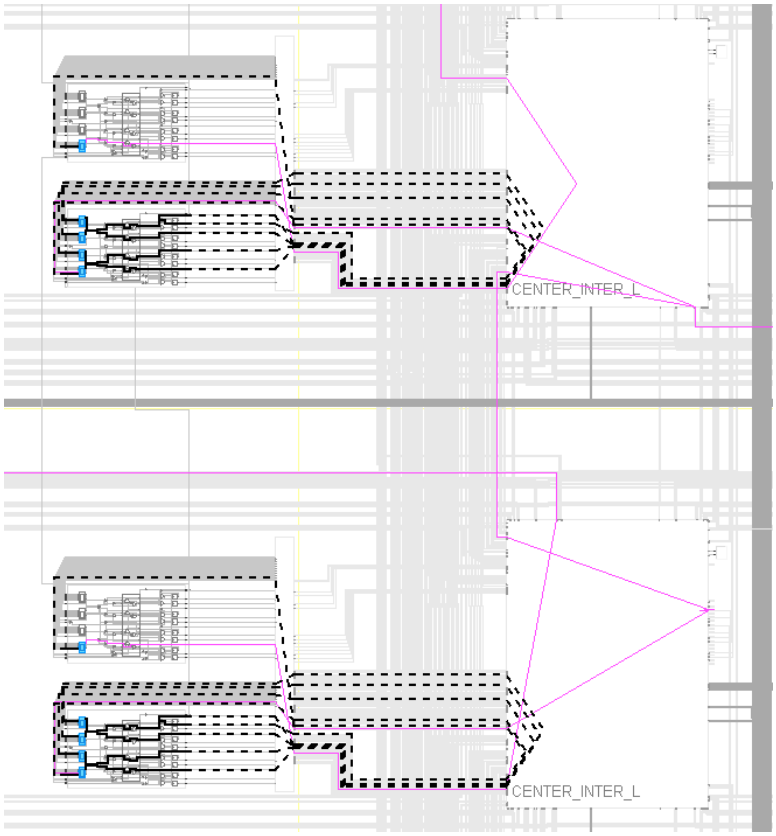


Figure 3·14: Two ROs with identical routing (highlighted by bold dotted lines). For better demonstration, RO_0 at the lower CLB is manually routed, whose routing is then cloned to RO_1 at the upper CLB.

where “x” and “y” are unique coordinates for different ROs.

The Arbiter PUF is a much more difficult case. For the RO PUF, each RO is more of a standalone system. As long as the routing among the five inverters of a RO is the same as the other ROs, the response should be able to reflect the manufacturing variations. However, in the Arbiter PUF, each pair of two MUXes of a stage are connected to both MUXes of the next stage, making the entire MUX chain an interconnected system. Under this circumstance, it is almost impossible to make exactly identical routing between two stages in FPGA. This is because the LUTs and their possible routing paths in an FPGA are not symmetrically allocated.

Briefly speaking, the placement and wiring of two neighboring slices are not mirrored. Some researchers claim that for this reason FPGA might not be an ideal platform for Arbiter PUF (Morozov et al., 2010).

Therefore a potential solution would be to make the routes as similar to each other as possible, in order to eliminate the timing differences from routing. Fig. 3·15 shows one example of such attempt.

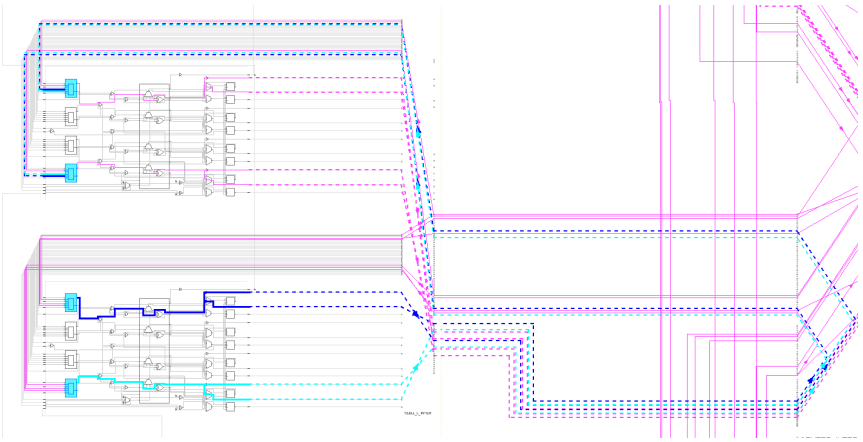


Figure 3·15: The routing highlighted in dotted cyan lines is from the lower MUX of a stage to both MUXes in the next stage, and the dotted blue lines are the route from the upper MUX to the next stage. The two routings are not exactly the same but are close enough to each other to minimize the impact of routing difference.

In addition, the RO PUF's routing can be easily carried out by acquiring one RO's routing and then duplicate to another using tcl commands or xdc constraints. But the Arbiter PUF's routing between stages has to be manually configured and fixed in order to achieve as much symmetry as possible. Although an FPGA provides many routing options from one BEL to another, the routing path cannot be "invented" by simply editing the xdc constraints. It has to be explored using the "Assign Routing Mode" (Xilinx, 2018a).

The "Assign Routing Mode" is a FPGA editing mode which allows users to apply their own configurations post implementation. However from one cell to another there

are only limited resources (paths) for routing. Users will need to manually discover the best BEL, pin, and route for a cell within this limited resource.

We will use the notations in Fig. 3·16 for better illustration.

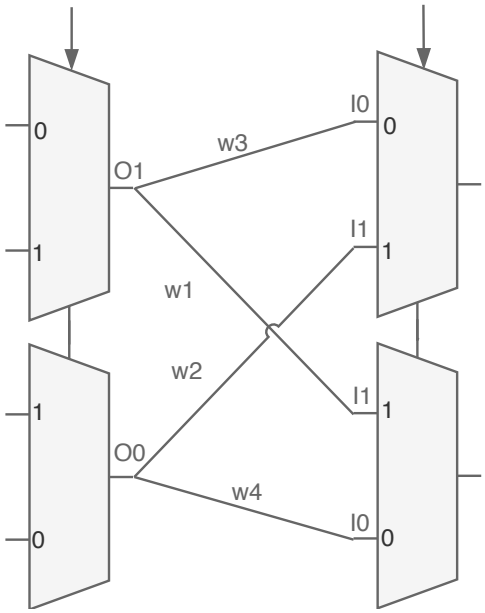


Figure 3·16: The routing of w1 should be the same as w2 in terms of length and path shape, and w3 the same as w4. I1 of the upper MUX should be the same pin as the I1 of the lower MUX, similarly to I0.

The instructions below can be followed to manually configure and fix the routing from one cell to another.

1. First, the placement of two stages of MUXes (4 MUXes) can be fixed symmetrically by the set_property BEL and set_property LOC commands using xdc constraints. For now the pins will be left unfixed to allow the routing with more flexibility.

2. Then, after running the implementation, in the "Device" tab by right clicking on the wires sprung from pin O1, a menu will pop up from which "Unroute" can be selected to cancel the current automatic routing of the wire. By right

clicking on O1 again we can select “Enter assign routing mode”, which enables us to manually route O1 to the two MUXes of the next stage.

3. After choosing the “Enter assign routing mode”, we will be asked if we want to set the routing target/destination or not. In Fig. 3·16 it refers to pin I0 of the lower MUX and pin I1 of the upper MUX in stage 2. For more routing flexibility, “No load” should be chosen to explore all possible options of routing to any cell and any pin.

4. Once “No load” is selected, then a window will pop up to let the user choose where to start routing from. Usually the first node in the list can be selected. Then a menu named “Routing Assignment” will appear on the right hand side of the device window. By clicking on the first item (node) under “Assigned Nodes”, the next possible nodes and their connections will show up in dotted lines, from which we can set the routing path node by node, and finally to the target cell. If a node and its route is not preferred, one can also right click on it in the “Assigned Nodes” window to remove them from the path.

5. When the wire is finally routed to the target MUX, all the newly routed path will be displayed in bold. This accomplishes the routing of w1 from O1 to I0 as shown in Fig. 3·18.

6. Then w2 should be routed in a similar way to minimize the delay difference caused by unsymmetrical routing. In addition, once the routing of w1 is determined, the pin for I0 is also determined. Therefore w2 should also be routed to the same pin of the upper MUX’s I0 as shown in Fig. 3·16.

7. After w1, w2, w3, w4 are all routed, their routing path can be read out by:

```
get_property ROUTE [get_nets wirename]
```

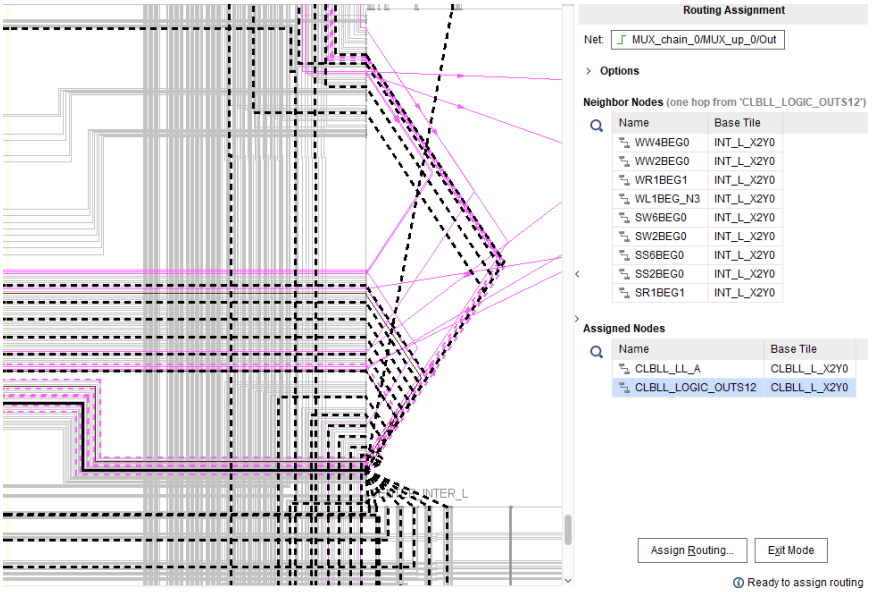


Figure 3·17: All the dotted black lines represent the next possible nodes following the node "CLBLL_LOGIC_OUTS12", which is highlighted in the "Assigned Nodes" menu. By selecting the nodes one by one, a user can route the output of a MUX to the inputs of the two MUXes in the following stage until he/she finds the desired path.

which will be a list of all the nodes in its path:

```
{ CLBLL\_LL\_A { CLBLL\_LL\_AMUX CLBLL\_LOGIC\_OUTS20 IMUX\_L36 CLBLL\_L\_D2 }
    CLBLL\_LOGIC\_OUTS12 IMUX\_L0 CLBLL\_L\_A3 }
```

In this way a designer will be able to explore and discover the best placements, pins, and routes for his/her elements in the Arbiter PUF. □

3.6 Multi-identity PUF (Mi-PUF)

In the previous section the concepts and design of RO and Arbiter PUFs have been introduced. Based on these two basic PUFs we now discuss the multi-identity PUF (Mi-PUF) (Bu and Kinsy, 2018a).

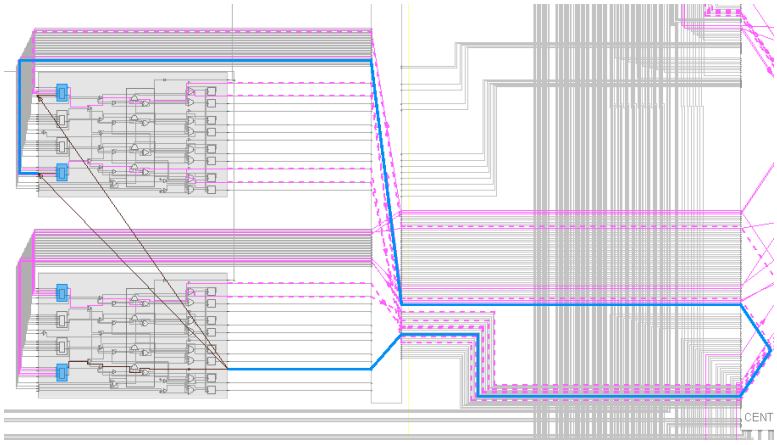


Figure 3·18: The w1 is successfully routed to the input pin I0 of a MUX in the next stage via the bold blue path.

3.6.1 Authentication Protocol of Mi-PUF

For a conventional PUF, no matter how many challenge and response pairs (CRPs) it has, it can only be identified and authenticated for one identity. However, a Mi-PUF can be identified and authenticated for multiple identities through an extra input called identity selection *Iden-Sel*. In addition, although a Mi-PUF remains a single piece of hardware, with different *Iden-Sel* inputs, each identity has a distinct CRP behavior from another. We introduce the new notations:

- CHL_k: the k^{th} challenge to a Mi-PUF;
- D_{i_j}: the j^{th} identity of the Mi-PUF indexed by i;
- $|ID|$: the total number of a Mi-PUF's identities;
- $|id|$: the number of identities a verifier can verify on a Mi-PUF, $|id| \leq |ID|$;
- RSP_{k_j}: the response of CHL_k under identity D_{i_j} of the Mi-PUF.

Protocol 3.6.1. The Mi-PUF's authentication procedure for multiple identities on a single device is as follows:

1. Before a Mi-PUF is delivered to its owner D_i, the manufacturer will challenge it with multiple challenges $\{CHL_0, CHL_1, \cdots, CHL_k, \cdots\}$ for $|ID|$ rounds under identities $\{D_{i_0}, D_{i_1}, \cdots, D_{i_{|ID-1|}}\}$. The responses are stored under each identity's CRP set;

2. A verifier acquires $|id|$ number of CRP sets from the manufacturer in a trusted way, where $|id| \leq |ID|$;

3. When the owner claims to be D_{i_j}, it needs to be verified by taking the verifier's CHL_k to its Mi-PUF;

4. When the Mi-PUF returns a response to the verifier, this response will be compared with the pre-stored RSP_{k_j} under identity D_{i_j} to check its validity. If it matches with the verifier's record, then the Mi-PUF owner is legitimately associated with D_{i_j}. The verifier can repeat the steps above to validate other identities as well.

3.6.2 Design and Implementation of Mi-PUF

In a Mi-PUF, three functional blocks are integrated: the ID box, Strict Avalanche Criterion (SAC) network, and First Order Reed-Muller (FORM) encoder.

Fig. 3·19 and 3·20 illustrate the basic element of the RO-based and Arbiter-based Mi-PUFs. The ID Box is installed between every two neighboring stages. The SAC and FORM encoder transform the *Iden-Sel* for security purposes.

3.6.3 ID Box

The ID box manifests the personality variation of Mi-PUF by altering the PUF circuit. It consists of two inverters and one 2-to-1 MUX as shown in Fig. 3·19 and 3·20. Each ID box takes in one bit of the transformed *Iden-Sel* input, and will affect the timing but not the value of the propagated signal. With different inputs to the ID Box, the RO- and Arbiter-based Mi-PUFs will function under various timing characteristics, thus behaving differently under the change of *Iden-Sel*.

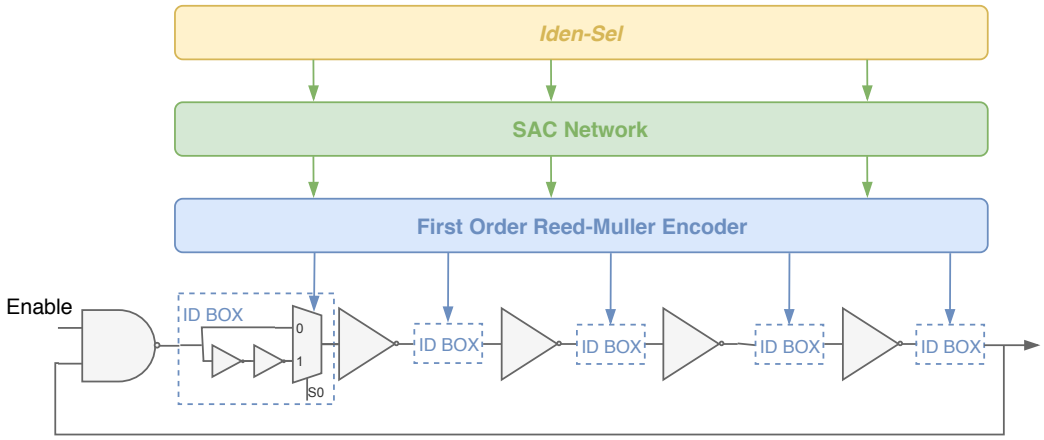


Figure 3·19: The upgraded RO for the RO-based Mi-PUF.

If more identities are needed (especially for weak PUFs such as RO PUF), the ID box can further evolve in at least two ways as shown in Fig. 3·21: 1) by fitting in multiple ID boxes between two stages; 2) by adding more choices of timing routes to the MUX. Both points are able to provide a great number of additional circuit delay characteristics to the Mi-PUF.

3.6.4 Strict Avalanche Criterion Network

On receiving an input of *Iden-Sel*, it will be first transformed by a Strict Avalanche Criterion (SAC) network, which is also used in the design of lightweight secure PUFs (Majzoobi et al., 2008). In a SAC network, whenever a single input bit is flipped, each output bit should have a probability of 0.5 to flip. The introduction of the SAC network is to increase the unpredictability and diversity a Mi-PUF circuit generated by a given *Iden-Sel*. Thus every two *Iden-Sel* inputs with a small Hamming distance will not result in similar circuits. This helps to prevent learning attacks across different identities.

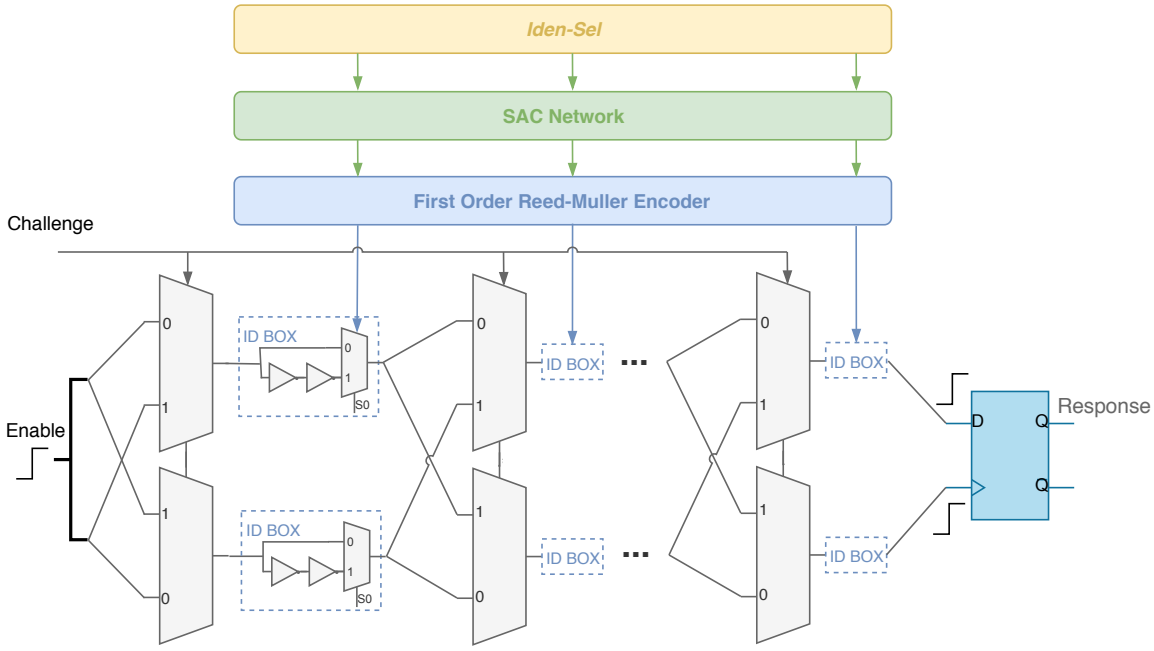


Figure 3·20: The upgraded MUX chain for the Arbiter-based Mi-PUF.

3.6.5 First Order Reed-Muller Encoder

Encoding schemes using error control codes (Bu et al., 2018d) are usually adopted to address the issue of side channel attacks on PUFs (Mahmoud et al., 2013). In the Mi-PUF design the First Order Reed-Muller (FORM) encoder is utilized to address this issue. An N-bit FORM codeword can be generated by check matrix $M = [\frac{M_1}{M_0}]$, where M_1 is a single row of all 1's, and the columns of M_0 consist of all different vectors of $\lceil \log_2 N \rceil$ bits. One important attribution of FORM codes is to ensure equal weights and uniformity of 1's and 0's in all its outputs (the vectors of all 1's and all 0's are excluded). In this way there are always half of ID boxes turned on in the slower route (port 1 of the MUX in the ID Box), and half of them in the faster route (port 0 of the MUX), which makes the power analysis on different personalities harder for attackers. The price to pay is the encoding redundancy.

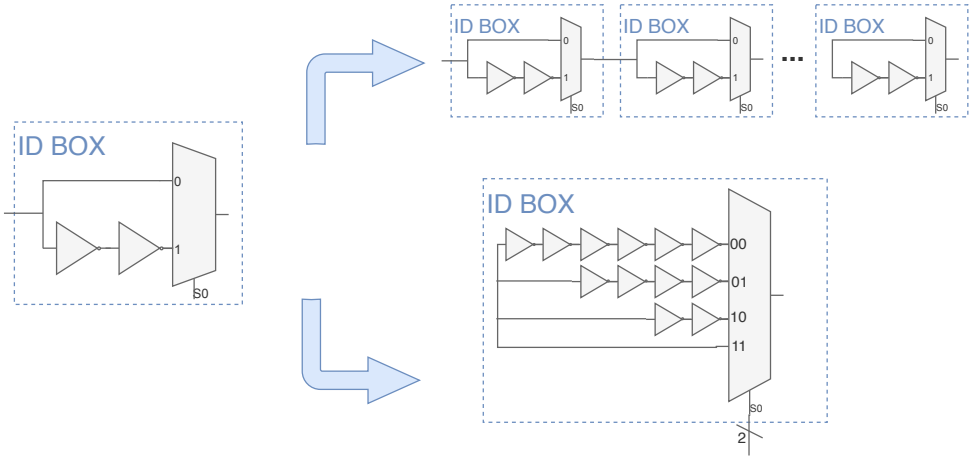


Figure 3·21: In the basic design of an ID Box, each box possesses two identities, and so the five ID Boxes in Fig. 3·19 provide 2^5 identities in total. Suppose it is replaced by h 2-to-1 ID boxes or one h-to-1 ID box, then the number of identities will increase to 2^{5h}.

3.7 Design and Implementation Evaluation

In this section, we evaluate the design and implementation of Mi-PUF in terms of (1) response sizes, (2) uniformity, (3) uniqueness of the identities using intra-board/chip and inter-board/chip Hamming distances.

3.7.1 $|CRP|$ and $|RSP|$ Set Sizes

For a five-stage RO PUF based Mi-PUF (cf. Fig. 3·19), it is fair to assume that there are m ROs in an RO group and r of such groups. Since the number of identities will be $|ID| = 2^5$, the total number of CRPs is:

$$|CRP| = 2^5 \cdot \binom{m}{2}, \tag{3.6}$$

which is also the total number of unique responses $|RSP|$.

For an Arbiter PUF based Mi-PUF in Fig. 3·19, the assumption is c MUX pairs in a MUX chain and r of such chains. The number of identities is $|ID| = 2^c$ and the total number of CRPs and unique responses are:

$$|CRP| = 2^c \cdot 2^c = 2^{2c}, \tag{3.7}$$

(Gassend et al., 2008) proposed a controlled PUF (CPUF) which introduced the concept of a personality input to increase the range of the conventional PUF. The differences between a CPUF and a Mi-PUF are:

1. A CPUF does not change the conventional PUF's structure. It remains the same piece of hardware while using hash to combine the personality input and challenge into a new CHL. As for the Mi-PUF, each identity is linked to a unique PUF circuit and signal route.

2. A CPUF does not increase the number of unique responses, while Mi-PUF does. Mi-PUF also provides more $|CRP|$ than the CPUF under the same settings.

Table 3.1 shows the differences among the proposed Mi-PUF, conventional PUF, and CPUF on the $|CRP|$ size and unique response set size $|RSP|$.

Table 3.1: $|CRP|$ and $|RSP|$ Comparison

PUF	Proposed Mi-PUF		conventional PUF		CPUF	
Type	$\|CRP\|$	$\|RSP\|$	$\|CRP\|$	$\|RSP\|$	$\|CRP\|$	$\|RSP\|$
RO	$2^5 \cdot \binom{m}{2}$	$2^5 \cdot \binom{m}{2}$	$\binom{m}{2}$	$\binom{m}{2}$	$a \cdot \binom{m}{2}$	$\binom{m}{2}$
Arbiter	2^{2c}	2^{2c}	2^c	2^c	$a \cdot 2^c$	2^c

[I] a the constant is the number of personalities.

[II] The proposed Mi-PUF has the largest $|CRP|$ and $|RSP|$, CPUF the second, and the conventional PUF the smallest. Posessing more responses can help the PUF be more resistant to modeling attacks.

3.7.2 Uniformity

Under the uniformity testing shown in Fig. 3·22, we examine the PUF's responses in terms of their bit vector balance between 0's and 1's. The ideal is 50-50 ratio. For a

given response, uniformity is calculated by:

$$\text{Uniformity} = \frac{1}{r}\sum_{r}^{j=1} RSP(j) \cdot 100\%$$

where $RSP(j)$ is the j^{th} bit of the response. We examine 32 identities' uniformity in each of the five Mi-PUF sizes with $r \in \{8, 16, 32, 96, 128\}$.

Figure 3·22: The uniformity of both RO- and Arbiter-based Mi-PUFs are around 35% to 50% across the 32 identities. The larger is the PUF size, the better the uniformity.

3.7.3 Uniqueness

Uniqueness is one of the most important parameters of a PUF and is evaluated using the average *Hamming Distances* (HD) of the PUF's responses. We provide two types of uniqueness for the Mi-PUF's evaluation in Table 3.2: 1) the "PUF uniqueness" across 12 FPGA boards (inter-board uniqueness under the same *Iden-*

Sel), and 2) the "identity uniqueness" across $|id| = 32$ identities on one FPGA (intra-board uniqueness under different *Iden-Sel*). Both are calculated based on the average Hamming Distance (HD) of their responses under the same CHL:

$$\text{Uniqueness} = \frac{2}{l(l-1)} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} \frac{HD(RSP_i, RSP_j)}{r} \cdot 100\%$$

where l is the number of boards for the inter-board uniqueness, and number of identities for the intra-board.

Table 3.2: Uniqueness Evaluation

	Inter-board		Intra-board	
	RO-based	Arbiter-based	RO-based	Arbiter-based
Without Hash	33.81%	19.13%	25.40 %	23.09 %
With Hash	49.27%	49.31%	49.30%	49.29%

[I] It can be seen that even without hashing the Mi-PUF response, the identically placed and routed Mi-PUFs already achieve an acceptable uniqueness (50% being the ideal).

3.8 Conclusion

In this chapter we explore the design and implementation of delay-based PUFs on FPGA, which will serve as the hardware root-of-trust for the proposed architectural model to provide authentication, identification, key storage functionalities.

We propose a novel type of PUF named multi-identity PUF (Mi-PUF). Unlike ordinary PUFs which attest only one legal identity of an element, the Mi-PUF is able to demonstrate multiple legitimate identities from one element. With this technique multiple unique IDs can be assigned to a PE upon its different activities (e.g., memory accesses or island memberships).

For implementation, we point out several common errors that designers may encounter in using this new platform, and provided solutions to them. We also suggest an improved design for the RO PUF, and instructions on how to discover the quasi-symmetric routing for the Arbiter PUF.

Chapter 4

Towards Programmable All-Digital True Random Number Generators

4.1 Introduction

Modern cryptographic approaches like encryption and authentication can achieve stronger security guarantees by making their algorithms public, instead of relying on obscurity as in early cryptographic algorithms. The open nature of these cryptographic mechanisms anchors the actual security of the system in the strength of secret vectors (Vassilev and Hall, 2014) (NIST, 2012), such as keys, obfuscation masks, or one-time pads etc. A secret key or vector is considered secure, if it is hard enough to guess/predict by any malicious parties. In other words, it needs to be random, and preferably with high physical entropy.

For this very reason, true random number generators (TRNG) have become an important category of cryptographic primitives. In this dissertation, it serves as a critical component to construct the key pairs, generate random vectors, and sampling small errors for cryptographic primitives such as the public-key cryptosystem (PKC) or oblivious transfer (OT).

On the procedure of making a TRNG, we also notice that for different application purposes, various levels of randomness are usually required. For example, in order to facilitate tasks such as sampling or re-ordering, true random numbers of moderate quality are acceptable. However, random masking used to protect a system against

cold boot attacks, requires a large amount of random bits in a short period of time. For cryptographic applications relying heavily on secret keys, such as data encryption, high entropy and high quality random bits are generally desired for strong keys.

Usually, for different applications with various demands on random bits, different designs of RNGs has to be adopted. There are also cases when within a single system, multiple modules need random numbers with different requirements, e.g., high throughput for random masking, low power for frequent nonces, and high quality for secret keys etc. Therefore, in order to tackle this issue, we propose a design of programmable multi-purpose TRNGs to satisfy these needs altogether. The major contributions of this design are:

1. It provides a real time programmable framework of the TRNG. Users are able to tune a number of parameters in order to generate a random sequence fitting a specific demand: entropy, throughput, power, and quality etc.;

2. The tuning of the TRNG's parameters also enables a cost-performance trade-off, where less energy is consumed with lower quality or throughput demand, and high quality or throughput will be available if more energy is allowed;

3. It achieves better performance than its competitors in terms of the randomness quality, throughput, and energy cost per bit, when configured to work in the corresponding mode.

The proposed programmable multi-purpose TRNG uses chaotic maps to further amplify the randomness of the seeds from its physical entropy source. Chaotic maps have the property that once a proper set of system parameters are selected, the output of the function will be highly sensitive to their initial state. This divergence property fits well in the concept of random number generation, which has become a popular field where chaotic systems are applied to. In the past decade, various types

of pseudo or true random number generators based on chaotic maps (e.g., logistic, Bernoulli shift, and dissipative quantum maps etc.) have been proposed (Lynnyk et al., 2015), (Kim et al., 2017), (François et al., 2014), (Akhshani et al., 2014), (de la Fraga et al., 2017), (Wang et al., 2016a), (Özkaynak, 2014). These works have presented approaches to design RNGs successfully passing the National Institute of Standards and Technology (NIST) random bit test (Bassham III et al., 2010). In this chapter, these works will serve as the comparative designs of the proposed design using Lorenz chaotic maps.

Besides the chaotic map-based RNGs, a more general comparison is also made between the proposed work and a number of representative TRNGs of various types (Kohlbrenner and Gaj, 2004), (Tsoi et al., 2003), (Danger et al., 2009), (Kwok and Lam, 2006), (Wieczorek and Golofit, 2014), (Sunar et al., 2007), (Wang et al., 2009), (Martin et al., 2018), (Cret et al., 2008), in terms of output quality, throughput, and energy cost etc.

The rest of the chapter is organized as follows. Section 4.2 lists several possible applications of the proposed TRNG, as well as the preliminaries of the Lorenz chaotic maps. Section 4.3 introduces the architecture of the programmable TRNG. Section 4.4 is on the FPGA-based digital physical entropy source. Section 4.5 explains the methodology and experiments on parameter tuning, in order to present the programmable characteristic of the proposed multi-purpose TRNG. Finally, a comprehensive comparison is made between the proposed work and other RNGs of fine quality. Section 4.6 concludes the chapter.

It is notable that in previous chapters, the designs usually strictly land on theoretical definitions and proofs. In this chapter, however, the design of TRNG requires extensive experiments in order to derive the best configuration. This is due to the nature of unpredictability and unstableness in a TRNG who relies on physical en-

tropy. Thus, in this chapter a large amount of data were collected and presented, from which we are able to suggest a set of proper parameters to tune the proposed TRNG for different purposes.

4.2 Motivation and Preliminaries

4.2.1 Possible Applications of the Proposed Design

As mentioned in the Introduction Section, the proposed design aims to provide a programmable TRNG, in order to serve various demands on random bits in a system.

For example, in the proposed new architectural model where Ring-Learning With Errors (R-LWE) public-key encryption is often utilized, the public key $\{a, b\}$ can be generated by a uniformly random vector $a \in R_q^n$ and $b = a \cdot s + e$, where $s, e \leftarrow \mathcal{X}$ the Gaussian noise sampling. In this key generation procedure, a should come from the high quality output of the TRNG, and the error e can be generated in a moderate quality mode due to its smaller value range limited by the error rate and Gaussian distribution. Similarly, in the OT scheme, the random vectors $\{r_0, r1, \cdots, r_{n-1}\}$ demand less in quality than the public key pairs.

In a nutshell, the multi-purpose primitive fits into systems characterized by Fig. 4·1.

The system in Fig. 4·1 consists of multiple modules $\{M_0 \cdots, M_i, \cdots, M_n\}$ with different properties. For example, M_0 carries less sensitive tasks, but is called frequently and thus consumes lots of random bits. In this case a TRNG with a small energy/bit cost and mediocre randomness quality will be sufficient. Meanwhile, M_n processes highly sensitive tasks and is activated less frequently. Therefore a high quality TRNG which is cryptographically secure will be the best option. For this case, it is reasonable to have a slightly larger energy/bit cost. Other modules in between have the similar trade-offs as shown in the figure.

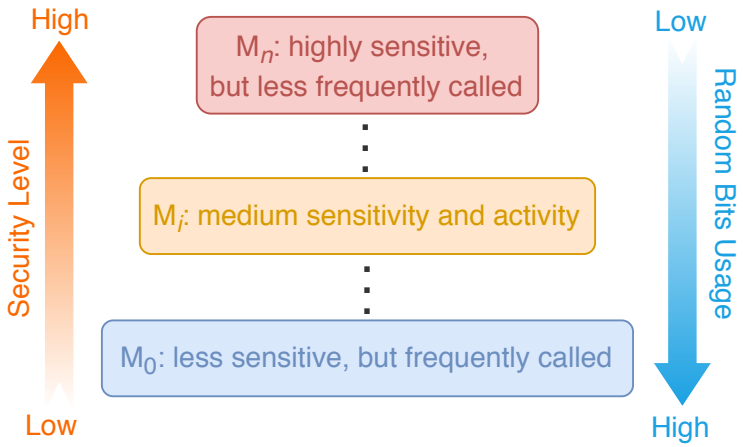


Figure 4·1: Different modules in the system above require different security levels and random bits usages. The random sequences for each module therefore have different characteristics.

The system designer can either equip each module with an ad hoc TRNG fitting its demand, or more wisely, have one universal TRNG which is able to satisfy all the demands through real time configuration. In this work, we construct a programmable multi-purpose TRNG for comprehensive systems like Fig. 4·1. Besides the examples listed previously, some other possible application scenarios are discussed below.

Information Storage Systems

A straightforward application is any information storage systems with different levels of sensitivity and data validity periods, such as a military intelligence program (MIP) or memory hierarchy.

There are data with short validity periods which get processed and dumped frequently (i.e., the less sensitive information processed daily in MIP, or data in L1 cache in a memory hierarchy). There are also data with longer validity periods which need to be protected with more security (i.e., highly sensitive information in MIP, or data in the memory module). There can also be cases in between. Such a system fits the model in Fig. 4·1 and can use a programmable TRNG to generate different levels of

random masks or keys for each scenario.

Distributed Systems

For an edge node in a distributed systems, its communication with other nodes in the same local group will be frequent but less sensitive, similar to the M_0 case in Fig. 4·1. Its interaction with the nodes in neighboring groups or middle stations will be less frequent, but probably with more sensitive information (the M_i case). And its communication with the cloud server might be the least frequent but require the highest security (the M_n case).

Randomness Functions in OS

Linux users often use `/dev/urandom` to generate random bits for most general purpose programs, and `/dev/random` for any highly confidential programs demanding more entropy and security. While `/dev/urandom` outputs with high throughput and never blocks (the M_0 case), `/dev/random` (the M_n case) frequently blocks whenever the entropy pool is empty (collected random bits from system's physical noise are used up).

Although regular computers can always feed the entropy pool with keyboard strokes or sound chip noises, it is not the case for web servers without those peripherals (Connolly, 2007), which becomes a potential vulnerability in their secure sockets layer (SSL) relying heavily on secret keys. While solutions such as `HAVEGE` (Seznec and Sendrier, 2003) by replacing Linux Kernel Module deals with the problem in software, a programmable multi-purpose TRNG can be a lightweight hardware fix.

4.2.2 Preliminaries of the Lorenz Chaotic Systems

Chaotic maps are a type of nonlinear and unpredictable systems which are highly sensitive to the initial condition. In such a system, any slight difference in the initial

state will produce rapid escalating and compounding variations in the system's future behavior. These phenomena capture the infinite complexity of the nature, which is often described by fractal mathematics and thus can serve as a proper candidate of randomness generation.

There are many types of chaotic systems, such as one-dimension (1D) logistic map, two-dimension (2D) Van der Pol system, and three-dimension (3D) Chua circuit etc. In this chapter we focus on the three dimensional Lorenz system. The Lorenz system was originally invented to describe and model the consequent bidirectional convection of thermally induced fluid, which is uniformly heated from below and cooled from above. However, because of its chaotic properties, it has also been used for a number of cryptographic purposes. The most common application is obfuscation such as block cipher (Jakimoski and Kocarev, 2001) and image encryption (Kwok and Tang, 2007). Another popular usage is on key agreements (Guo and Chang, 2013). There also have been researches to take advantage of both divergence and convergence to achieve lightweight authentication (Bu et al., 2018a).

To better facilitate the following presentation, the following notations are suggested:

- α, β, γ: the parameters of Lorenz system's functions;
- $p_n = (x, y, z)$: the output point of a three-dimension (3D) Lorenz function. n stands for the number of iterations a Lorenz function has been run before generating such output, and (x, y, z) the coordinates;
- $LF_i(p_{0\text{-}i}, n)$: the i^{th} Lorenz function characterized by $\{\alpha_i, \beta_i, \gamma_i\}$. With $p_{0\text{-}i}$ being the initial condition, $LF_i(p_{0\text{-}i}, n) = p_{n\text{-}i}$, where $p_{n\text{-}i} = (x_i, y_i, z_i)$ stands for the output of the i^{th} Lorenz function.

The discrete form of the Lorenz functions is given below:

$$\begin{cases} x_{n+1} = x_n + \alpha(x_n - y_n)\triangle t \\ y_{n+1} = y_n + (\gamma x_n - x_n z_n - y_n)\triangle t, \\ z_{n+1} = z_n + (x_n y_n - \beta z_n)\triangle t \end{cases} \tag{4.1}$$

where $\triangle t$ determines the resolution of the map.

Fig. 4·2 is shape of [Eq. 4.1] with system parameters $\alpha = 10, \beta = 2.6667, \gamma = 28$, which were the original values chosen only for fluid convection modeling. Obviously, other proper values can be selected to construct different Lorenz maps.

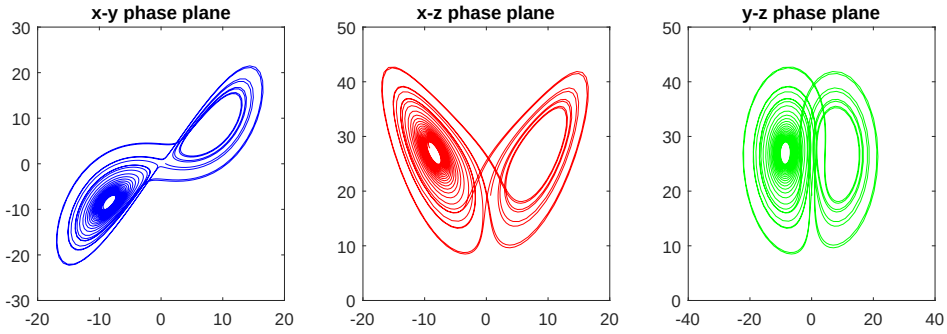


Figure 4·2: The 3d trajectory of a Lorenz system projected onto $x-y, x-z$ and $y-z$ planes, usually in a butterfly or "8" pattern.

The major properties of Lorenz functions are:

1. *Stationary points*: In [Eq. 4.1], when $\gamma > 1$, there are two distinct stationary points, which are:

$$C1, C2 = (\pm\sqrt{\beta(\gamma-1)}, \pm\sqrt{\beta(\gamma-1)}, \gamma - 1) \tag{4.2}$$

Although $C1$ and $C2$ are not physically on the trajectory, they serve as the *attractors* to balance out the initial transients, and drive the system towards its typical behavior.

2. *Convergence*: The attractors bring in the convergence property of a chaotic system. In other words, even if the initial state p_0 is not a point on the trajectory, it will soon be converged to the orbit within limited iterations. In addition, although the timing of a point appearing on the trajectory is highly unpredictable, over time the points are all conformed to the butterfly pattern statistically.

   The convergence attribution can be described by Hausdorff dimension $dim_H K$, which is bounded by (Pogromsky et al.,):

$$dim_H K \leq 3 - \frac{2(\alpha + \beta + 1)}{\alpha + 1 + \sqrt{(\alpha - 1)^2 + 4\gamma\alpha}} \tag{4.3}$$

3. *Divergence*: The key property leveraged for the construction of the proposed TRNG. Intuitively, the divergence comes from the high randomness of the location and timing that a point $p_n = (x, y, z)$ appears on a 3D Lorenz map. With a tiny variation of the initial condition p_0, after n iterations the final output p_n will be largely deviated. Theoretically speaking, Lyapunov exponent can be used to measure the rate of divergence of a chaotic system:

$$|\delta(p)| \approx |\delta(0)|e^{\lambda p}, \tag{4.4}$$

   where for a trajectory $T(p)$'s nearby orbit $T(p) + \delta(p)$, $\delta(p)$ is a vector with infinitesimal initial length. The maximal λ is known to be approximately 0.9056.

4.3 The Proposed Programmable True Random Number Generator

In this section we will introduce the architecture of the proposed programmable multi-purpose true random number generator (TRNG). First, a digital physical entropy source named Asynchronous STopwatch-controlled Ring Oscillator (ASTRO) pro-

vides true random seeds for the TRNG. Then a group of Lorenz functions work as randomness amplifiers of the seeds. For the randomness amplifiers we also have the choices of using ciphers or hash functions. We choose chaotic functions for their simple implementation and high throughput properties. Later we show that the design passes the NIST SP 800-90A test which is a standard of cryptographically secure random number generators.

The outputs of the Lorenz functions are quantified and blended together to further improve their quality, before being sent out as the TRNG output. The architecture of the TRNG is shown in Fig. 4·3.

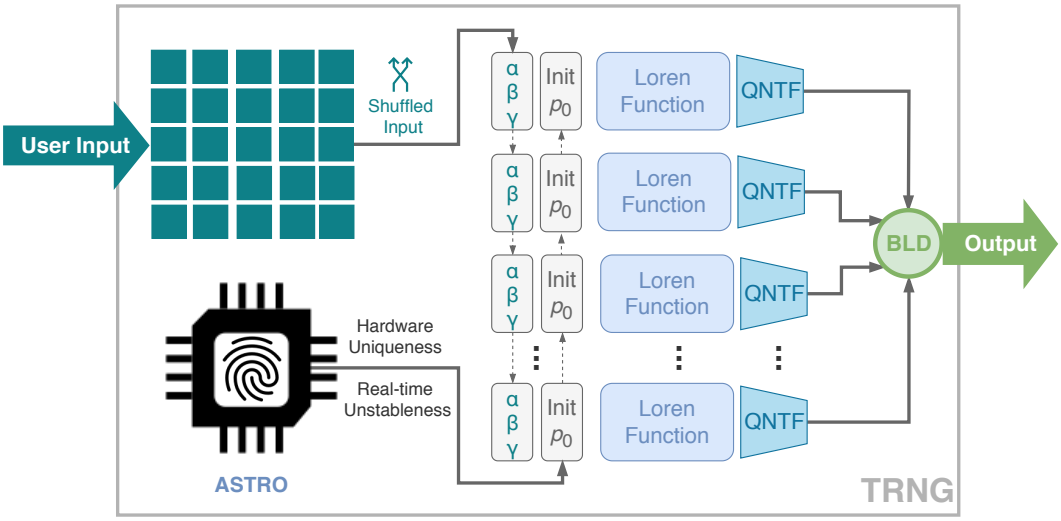


Figure 4·3: The inputs ($\{\alpha, \beta, \gamma\}$ and p_0) of the Lorenz Functions group are pre-processed by the SAC network and the ASTRO. Its outputs are processed by the QNTF and BLD modules. The architecture also features six customizable parameters for the tuning of the cost-performance trade-off.

In this design, all the vectors ($\{\alpha, \beta, \gamma\}$ and $\{x, y, z\}$) in the proposed TRNG are 64-bit, where the 8 most significant bits (MSBs) are the integer part, and the 56 least significant bits (LSBs) decimal.

There are six user inputs to program and customize the TRNG to the desired working mode (e.g., high physical entropy, high quality, high throughput, energy

saving etc.) as shown in Fig. 4·4, which will be introduced briefly in subsections 4.3.1 to 4.3.5, and explored in details with experimental data in Section 4.5.

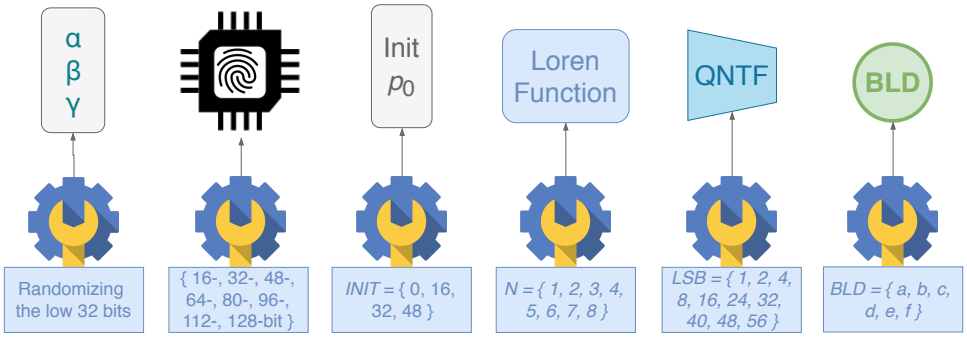


Figure 4·4: The six user inputs of the TRNG to program it to the desired setting for different purposes.

4.3.1 The SAC Network and Its Configuration of $\{\alpha, \beta, \gamma\}$

The proposed TRNG supports optional user inputs (any user-designated information source) which configures $\{\alpha, \beta, \gamma\}$ of the Lorenz functions. Although the user input is not mandatory since all $\{\alpha, \beta, \gamma\}$ can be preset, this configuration enables more diversity in the entire system.

According to [Eq. 4.2], the change of $\{\alpha, \beta, \gamma\}$ will result in the re-locating of the two attractors, which ultimately leads to a new chaotic map. Fig. 4·5 shows how the two attractors drift away with the change of $\{\alpha, \beta, \gamma\}$.

Given a user input (a binary vector), it will be transformed by a binary Strict Avalanche Criterion (SAC) network/matrix (Cao et al., 2015) for shuffling first, before being used for $\{\alpha, \beta, \gamma\}$ configuration. In a SAC network, whenever a single input bit is flipped, each output bit should have a probability of 0.5 to flip. Thus any two user inputs with a small difference, say only one bit off, will result in largely different configurations. For example, the S-Box in AES is a function satisfying SAC.

However, while the shuffled user input can be any arbitrary value, the Lorenz parameters $\{\alpha, \beta, \gamma\}$ cannot be arbitrarily configured. Otherwise the resulted trajec-

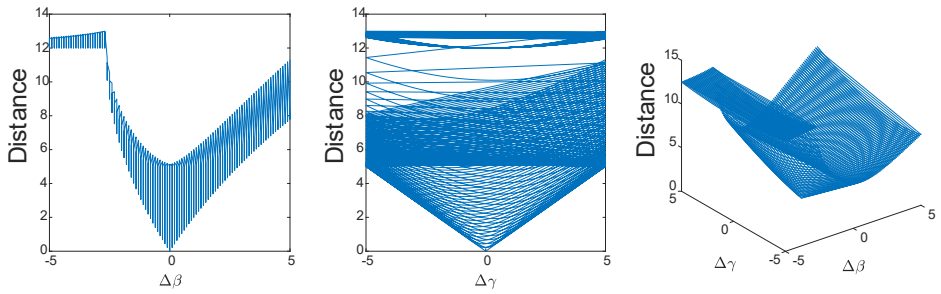


Figure 4·5: When β (left) or γ (middle) or both of them (right) fluctuates, the attractors will drift away according to the fluctuation magnitude. α on the other hand is related to the size of the trajectory. Every point with distance $\neq 0$ stands for a new chaotic map due to the change made to $\{\alpha, \beta, \gamma\}$.

tory may lose its chaotic property (Bu et al., 2018b). Fig. 4·6 shows the results of configuring all the 8 integer bits, all the 56 decimal bits, and only the last 32 decimal bits of a Lorenz map with the original parameters $\{\alpha = 10, \beta = 0.6, \gamma = 28\}$.

To be on the safe side, only the last 32 bits of all the $\{\alpha, \beta, \gamma\}$ are left for the shuffled input. It is also notable that due to the human-related nature of the user input, it does not necessarily provide security and unpredictability for the TRNG, which officially should be the job of ASTRO.

4.3.2 Asynchronous STopwatch-controlled Ring Oscillator (ASTRO)

The Asynchronous STopwatch-controlled Ring Oscillator (ASTRO) module serves as the physical entropy source to provide seeds of true randomness to the TRNG. ASTRO is a variant of ring oscillator-based physical entropy generator, as proposed by (Sunar et al., 2007) and (Wang et al., 2009). Besides providing high physical entropy (shown later in this chapter) as conventional RO does, ASTRO is able to achieve a larger throughput by its design.

ASTRO provides 16- to 128-bit true random seeds (per user's customization) for the initial condition p_0, which has three coordinates $\{x, y, z\}$. This feature fits perfectly into Lorenz function's divergence property, that a small variation in p_0 will

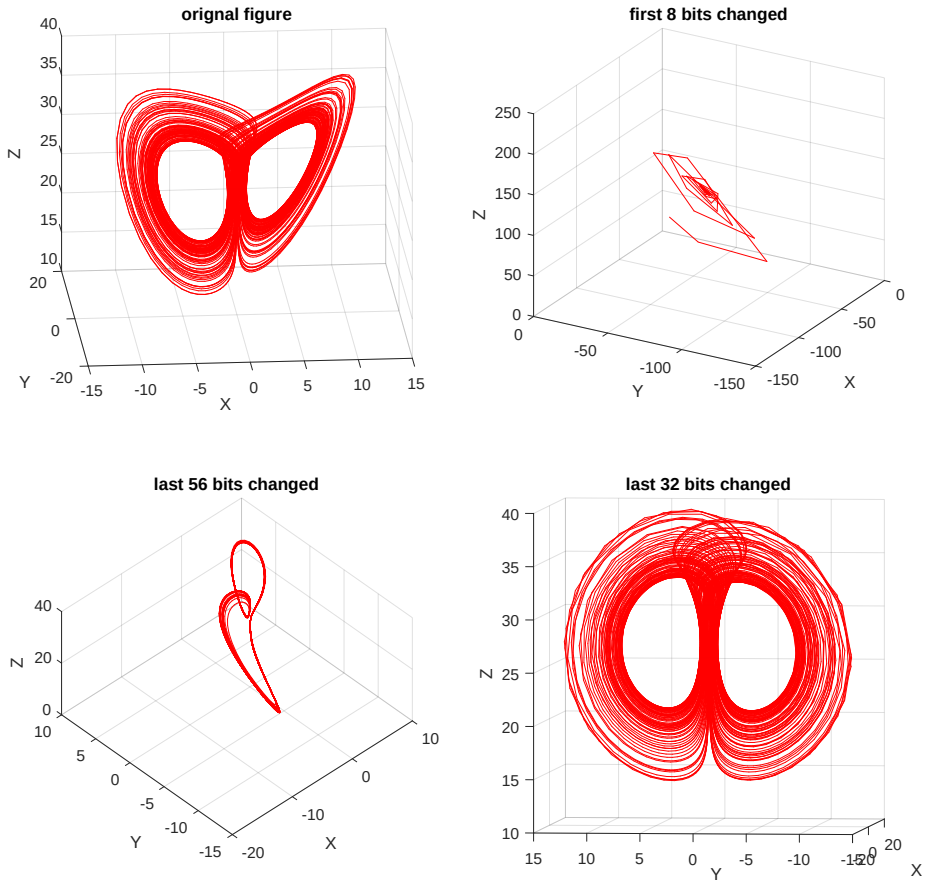


Figure 4·6: When the integer bits or all the 56 decimal bits are arbitrarily changed (up-right), the results trajectory could be no longer chaotic. Only the 48 LSBs or less (bottom-right for 32 bits) can be arbitrarily configured while maintaining the chaotic property.

lead to drastic deviation in p_n.

The ASTRO module manages a configurable parameter $INIT \in \{48, 32, 16, 0\}$, determining from which bit each coordinate is to be dynamically modified by ASTRO. The design and implementation details of ASTRO are presented in Section 4.4.

4.3.3 Lorenz Function Group

All Lorenz functions in the group have the 32 MSBs of their $\{\alpha_i, \beta_i, \gamma_i\}$ fixed with different values. Their 32 LSBs of $\{\alpha_i, \beta_i, \gamma_i\}$ and $\{p_{0\text{-}i}\}$ are arranged by the SAC and ASTRO. By [Eq. 4.1], a Lorenz function can be implemented on an FPGA by mainly

fixed-point adders and multipliers.

This module manages a configurable parameter $N \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, namely the number of Lorenz functions.

4.3.4 QNTF Module

The quantification (QNTF) module is in charge of truncating the outputs of the Lorenz functions. It has been proved (Lynnyk et al., 2015) that using the entire output vector as the random bit string will not pass the NIST test. This is because the MSBs of the output (especially the first 8) change very slowly as the iteration goes. Therefore a proper LSB should be chosen to maintain both the randomness and throughput.

Thus, the QNTF module manages a configurable parameter $LSB \in \{1, 2, 4, 8, 16, 24, 32, 40, 48, 56\}$, which determines the number of LSBs to use for the random string.

4.3.5 BLD Module

The blender (BLD) module blends all the $\{p_{n\text{-}i}\}$ from the Lorenz function group into the final TRNG output. The BLD module manages a configurable parameter *BLD* $\in$ {a, b, c, d, e, f}, selecting which of the following six formulas to incorporate all the QNTF-truncated $\{p_{n\text{-}i}\}$ into the final random string.

The six formulas below are designed to shuffle and combine the outputs of the N Lorenz functions by XORing, permuting, reversing, and interleaving in multiple ways. Indeed, more formulas can be explored and added to this module.

a. $(x_0 \oplus y_0 \oplus z_0)||\cdots||(x_{N-1} \oplus y_{N-1} \oplus z_{N-1})$;

b. $(x_0 \oplus \cdots \oplus x_{N-1})||(y_0 \oplus \cdots \oplus y_{N-1})||(z_0 \oplus \cdots \oplus z_{N-1})$;

c. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : z_i) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : x_i) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : y_i)$;

d. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : \overleftarrow{x_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : \overleftarrow{y_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : \overleftarrow{z_i})$;

e. $\bigoplus_{i=0}^{N-1}((i \text{ is even})?x_i : \overleftarrow{z_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?y_i : \overleftarrow{x_i}) \,||\, \bigoplus_{i=0}^{N-1}((i \text{ is even})?z_i : \overleftarrow{y_i})$;

f. $\bigoplus_{i=0}^{N-1}(x_i \oplus y_i \oplus z_i)$,

where $\oplus$ stands for bitwise XOR, $||$ concatenation, and $\leftarrow$ the bit order reverse operator.

4.4 The Physical Entropy Source: Asynchronous Stopwatch-controlled Ring Oscillator (ASTRO)

In this section we propose the physical entropy source of the TRNG named Asynchronous STopwatch-controlled Ring Oscillator (ASTRO). Unlike many chaotic map-based TRNGs using analog circuits as the physical entropy source, ASTRO can be conveniently programmed and instantiated as a digital circuit by Hardware Description Language (HDL) and implementation constraints on FPGAs.

4.4.1 The ASTRO Architecture

The micro-architecture of ASTRO is shown in Fig. 4·7. It consists of two five-stage ring oscillators (RO) and each clocks a counter. One serves as the other's stopwatch.

The ASTRO's true randomness comes from the RO's unpredictable frequency fluctuation. Due to the manufacturing variation of each gate, the two ROs will have different frequencies. In addition, according to the measurements, the RO frequency is highly sensitive to the surrounding environment and varies from time to time. Thus the timing of the faster RO's counter reaching overflow and triggering the "Stop" signal varies at each run, and the incrementing speed of the slower RO's counter also changes every time. These two interactive uncertainties together make it possible for true randomness.

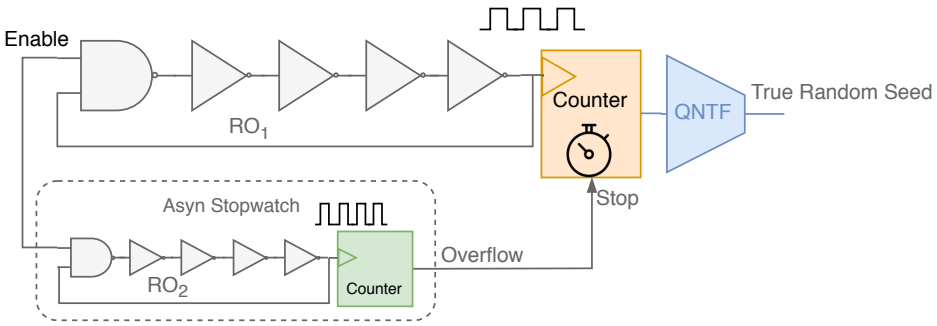


Figure 4·7: Whichever RO reaches to the counter overflow first, will send an asynchronous "Stop" signal to pause the other RO's counter.

The RO frequency is usually around 350 MHz and much higher than the 100 MHz FPGA clock. To clearly reflect the frequency difference (usually < 3MHz) between the two ROs, the counter size has to be no smaller than 28 bits. Through statistical analysis, the 12 MSBs of the slower RO's counter are relatively stable. However, the 16 LSBs always demonstrate adequate unpredictability, and can thus serve as a physical entropy source.

The proposed TRNG is equipped with eight ASTROs. Therefore it is able to output up to $8 \times 16 = 128$ bits of true random seeds. The entropy of each bit is calculated in Fig. 4·8 based on over 50,000 sets of ASTRO output data.

The total and average entropies produced by eight sizes of ASTRO outputs (from 16 to 128 bits) are shown in Fig. 4·9.

It is notable that in order to provide strong cryptographic keys, information security standards (Barker and Roginsky, 2011) require at least 112-bit of security strength from physical entropy (equivalent to seven ASTROs turned on). Since every individual ASTRO costs negligible power (0.005W), we suggest all eight ASTROs to be turned on for any security-oriented applications.

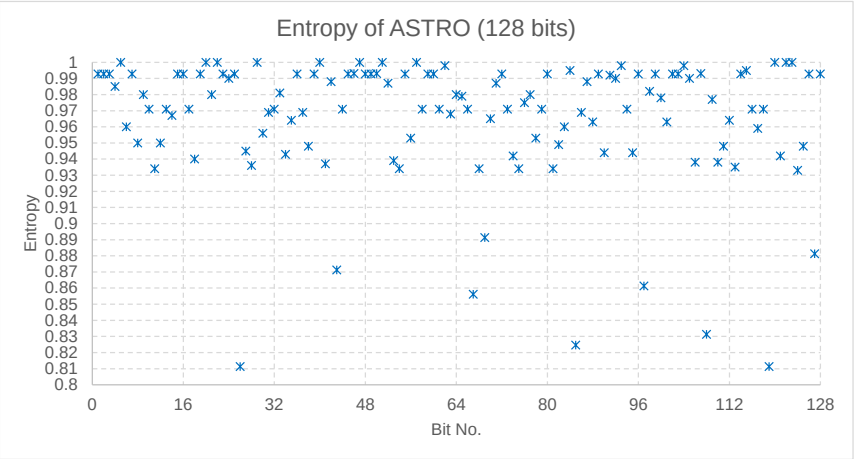


Figure 4·8: The entropies of most bits produced by ASTRO are located between the [0.93, 1] bit window, which is already in good quality. It can be further improved to a [0.997, 1] bit window by the proposed TRNG microarchitecture (cf. Fig. 4·17).

4.4.2 ASTRO FPGA Implementation

Once a piece of HDL code is written to generate an ASTRO, the two ROs' placement and routing need to be identically fixed. Otherwise the circuit timing produced by automated routing will dominate over the intrinsic hardware uniqueness between the two ROs (Bu and Kinsy, 2018b). In that case, ASTROs on different FPGAs will behave similarly, thus eliminating sufficient entropy production.

Hence, similar to the construction of physical unclonable functions (PUF), three constraints need to be set identically: relative placement, pins, and routing. For complete details one can refer to Section 3.5 in Chapter 3 on PUF.

Remark 4.4.1. It is notable that ASTRO is different from the physical unclonable functions (PUF). The later demands reproducibility for authentication purposes, while the former favors irreproducibility for randomness. Thus ASTRO has less restrictions and is much simpler to implement than PUF.

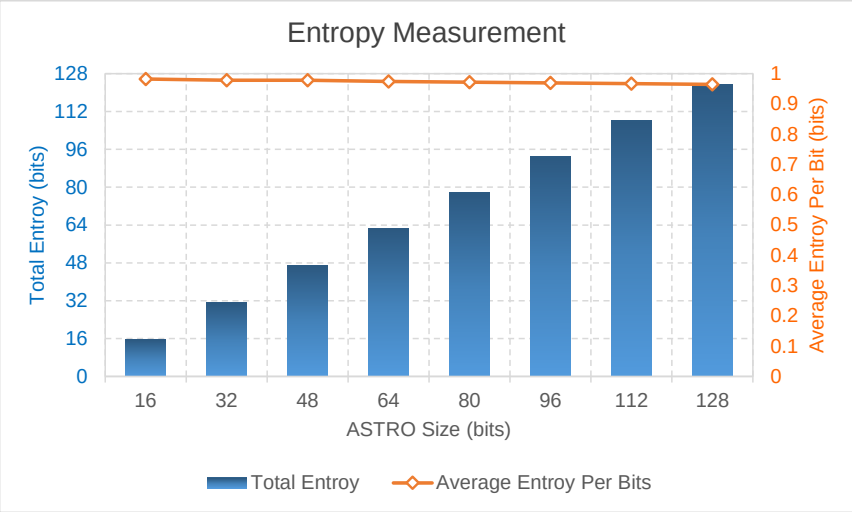


Figure 4·9: The total entropy in each case is very close to the output size. The average entropy provided by each bit of ASTRO is 0.964.

4.5 Programmability and Experiments

In this section, we study the behavior of the proposed TRNG by tuning its parameters in real time, which leads to its programmability to serve multiple purposes, such as producing high quality random sequences, outputting at large throughput, or functioning at an adjustable power range. Switching among these purposes also enables the trade-off between cost (energy/bit) and performance (randomness quality).

In addition, we evaluate the TRNG's output entropy and the key sensitivity. In the end, a comparison is made between the proposed TRNG and the existing chaotic map-based RNGs. We show that the proposed work is able to achieve higher output quality than all its competitors.

4.5.1 The Six Configurable Parameters

As shown in the TRNG's architecture in Fig. 4·7 and discussed in Section 4.3, the lower 32 bits of $\{\alpha, \beta, \gamma\}$ can be randomized by user input, and the physical entropy source (ASTROs) can also be adjusted by users from 16 to 128 bits. The larger the

physical entropy size, the more security it can provide when used for cryptographic purposes.

Besides these two, there are four configurable parameters *LSB, INIT, N*, and *BLD* to program the TRNG to work under different modes (high throughput, high quality, and low energy cost). The available values for the four parameters are:

Table 4.1: Configurable Parameters of the TRNG

LSB	$\{1, 2, 4, 8, 16, 24, 32, 40, 48, 56\}$					
INIT	$\{0, 16, 32, 48\}$					
N	$\{1, 2, 3, 4, 5, 6, 7, 8\}$					
BLD	a	b	c	d	e	f
Throughput	$N \cdot LSB$	$3LSB$	$3LSB$	$3LSB$	$3LSB$	LSB

[I] The unit of throughput is bits/cycle.

In this chapter, the randomness quality of all the possible parameter combinations are evaluated by their NIST test scores (p-values, goodness of fit). The NIST test consists of 15 sub-tests: AET, FBT, CST (forward (FW) and reverse (RV)), DFTT, FT, LCT, LROBT, NTMT, OTMT, RET, REVT, BMRT, RT, ST (1 and 2), MUST. Their detailed introduction can be found in (Bassham III et al., 2010). An exhaustive search is made for each parameter on its all possible combinations with other parameters. The search aims to discover two indicators on each parameter:

- Qualifying Rate: the ratio of passed tests over all tests;
- Sub-test Impact: how the change of that parameter affects the p-values of certain sub-tests.

LSB

It determines the number of least significant bits of the Lorenz functions' outputs to be kept for use.

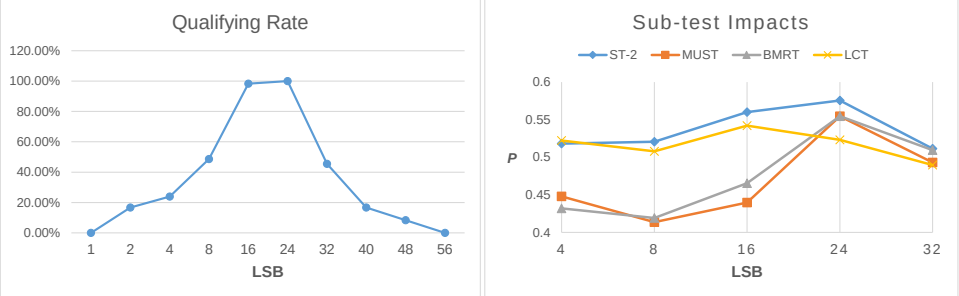


Figure 4·10: The qualifying rate is almost in a normal distribution under various *LSB* values.

As in Fig. 4·10, $\{4, 8, 16, 24, 32\}$ are acceptable values for *LSB* (the others have too low qualifying rates). Particularly, among the 17 sub-tests, $LSB = \{16, 24, 32\}$ have been found to have positive impacts in the ST-2, MUST, BMRT, and LCT.

INIT

It sets the ASTRO configuration window in each of p_0's three 64-bit coordinates.

As shown in Fig. 4·11, in terms of sub-test impacts, INIT = 32 shows its advantage over other values in many sub-tests: CST, DFTT, LCT, FT, ST-1, and ST-2.

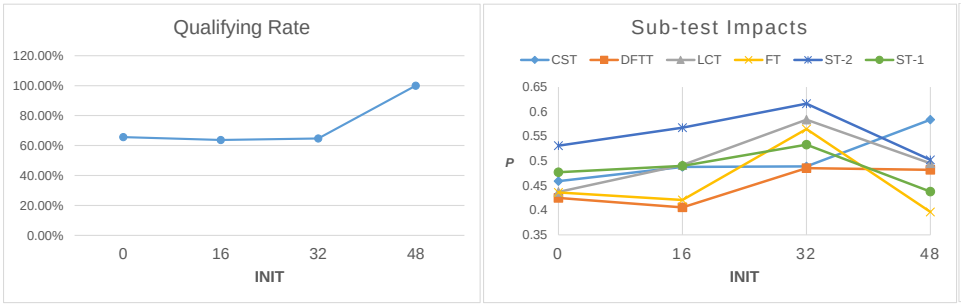


Figure 4·11: The qualifying rate when $INIT = 48$ reaches the highest. However, $INIT = 32$ is more influential in the sub-test scores.

N

It determines the number of Lorenz functions embedded in the TRNG.

As in Fig. 4·12, a larger N will be the proper choice if the design is random bits quality-oriented . However, the great N is, the more power-consuming the TRNG will be, due to the increasing number of multipliers.

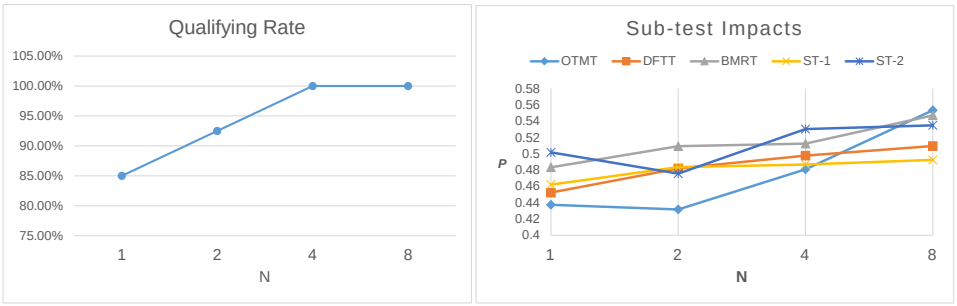


Figure 4·12: Both the charts show a trend that the greater N is, the better quality will be for the generated random bits.

BLD

It selects in which way the truncated outputs of Lorenz functions are to be mixed for the final TRNG output.

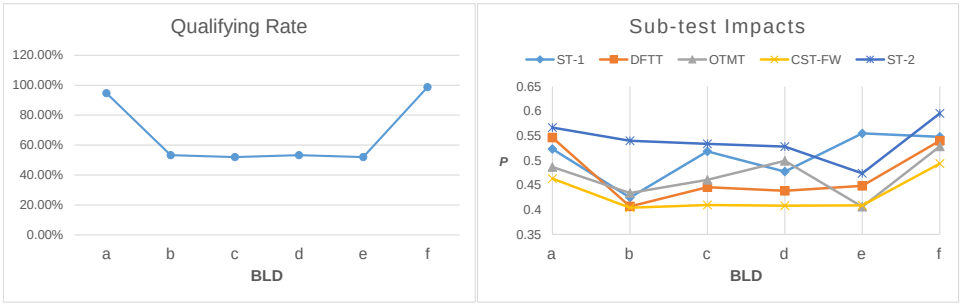


Figure 4·13: Both the qualifying rate and sub-test impacts show the superior performance when BLD = a, f, especially f.

Although BLD = f performs better than others, it produces the least amount of random bits in a circle (lowest throughput).

4.5.2 The High Throughput, High Quality, and Low Energy Cost Working Modes of the Proposed TRNG

The proposed TRNG's programmability to serve multi-purposes can be explained as below (more experimental data follow).

By Table 4.1, to work on the *high throughput* mode, *LSB*, *N*, *BLD* can be tuned to achieve various throughputs (bits/cycle): {16, 24, 32, $\cdots$, 128, 144, 168, 192, 224, 256}. By Figs. 4·10, 4·12, 4·13, a large *LSB* and N with $BLD = a$ will produce high throughput without sacrificing the randomness quality (p-values), but will consume more power.

To work on the *high quality* mode, by Figs. 4·10-4·13, the configuration needs to come out from $LSB \in \{16, 24\}$, $INIT \in \{32, 48\}$, $BLD \in \{a, f\}$, and a large N (leading to higher power consumption). Table 4.2 and Fig. 4·14 show the detailed NIST scores and their uniformity from the best configuration guided by such quality-oriented principles.

To work on the *low energy cost* mode, some Lorenz functions can be shut down (i.e., reducing N). This is because the most heavy-duty component in the design is the fixed-point multiplier of the Lorenz functions. However, by Fig. 4·12, reducing N will also reduce the randomness quality.

4.5.3 Cost-Performance Trade-off

As discussed in Section 4.5.1, the proposed multi-purpose TRNG is able to generate random sequences upon different demands. However, the power-, throughput-, and performance (quality)-oriented designs all come with some trade-offs, which are discussed as follows.

First, as stated previously, the power consumption is dominated by the fixed-point multipliers in the Lorenz functions. Other modules, such as the ASTRO, QNTF, and BLD, consume almost negligible power (< 0.1W) comparing with the Lorenz group.

Table 4.2: The NIST Scores of the Proposed TRNG under High Quality Mode ($LSB = 24$, $INIT = 32$, $N = 8$, BLD = f)

No.	Statistical Tests	Sequences with $p \geq 0.01$ (successfully passed)	Sequences with $p < 0.01$ (failed to pass)	Passing Rate	p-value (goodness of fit)
1	FT	991	9	99.10%	0.437274
2	FBT	990	10	99.00%	0.350485
3	CST-FW	995	5	99.50%	0.739918
	CST-RV	990	10	99.00%	0.275709
4	RT	993	7	99.30%	0.991468
5	LROBT	986	14	98.60%	0.739918
6	BMRT	991	9	99.10%	0.437274
7	DFTT	984	16	98.40%	0.637119
8	NTMT*	988.99	11.01	98.90%	0.452054
9	OTMT	987	13	98.70%	0.534146
10	MUST	1000	0	100.00%	0.911413
11	AET	989	11	98.90%	0.911413
12	RET (*1016 samples*)				
	(1) $x = -4$	124	3	97.64%	0.739918
	(2) $x = -3$	126	1	99.21%	0.350485
	(3) $x = -2$	126	1	99.21%	0.534146
	(4) $x = -1$	124	3	97.64%	0.739918
	(5) $x = 1$	126	1	99.21%	0.911413
	(6) $x = 2$	126	1	99.21%	0.739918
	(7) $x = 3$	122	5	96.06%	0.350485
	(8) $x = 4$	122	5	96.06%	0.213309
13	REVT (*2304 samples*)				
	(1) $x = -9$	128	0	100.00%	0.534146
	(2) $x = -8$	128	0	100.00%	0.350485
	(3) $x = -7$	127	1	99.22%	0.350485
	(4) $x = -6$	128	0	100.00%	0.213309
	(5) $x = -5$	128	0	100.00%	0.350485
	(6) $x = -4$	128	0	100.00%	0.350485
	(7) $x = -3$	128	0	100.00%	0.122325
	(8) $x = -2$	128	0	100.00%	0.911413
	(9) $x = -1$	127	1	99.22%	0.534146
	(10) $x = 1$	128	0	100.00%	0.739918
	(11) $x = 2$	126	2	98.44%	0.350485
	(12) $x = 3$	127	1	99.22%	0.350485
	(13) $x = 4$	127	1	99.22%	0.739918
	(14) $x = 5$	127	1	99.22%	0.534146
	(15) $x = 6$	126	2	98.44%	0.122325
	(16) $x = 7$	127	1	99.22%	0.991468
	(17) $x = 8$	128	0	100.00%	0.534146
	(18) $x = 9$	128	0	100.00%	0.911413
14	ST-1	989	11	98.90%	0.739918
	ST-2	994	6	99.40%	0.911413
15	LCT	983	17	98.30%	0.964295

[I] The test scores are calculated based on 1000 Sequences with bit stream length of 100,000 each.

[II] Averaged results are taken for the tests marked with * (i.e., the NTMT test).

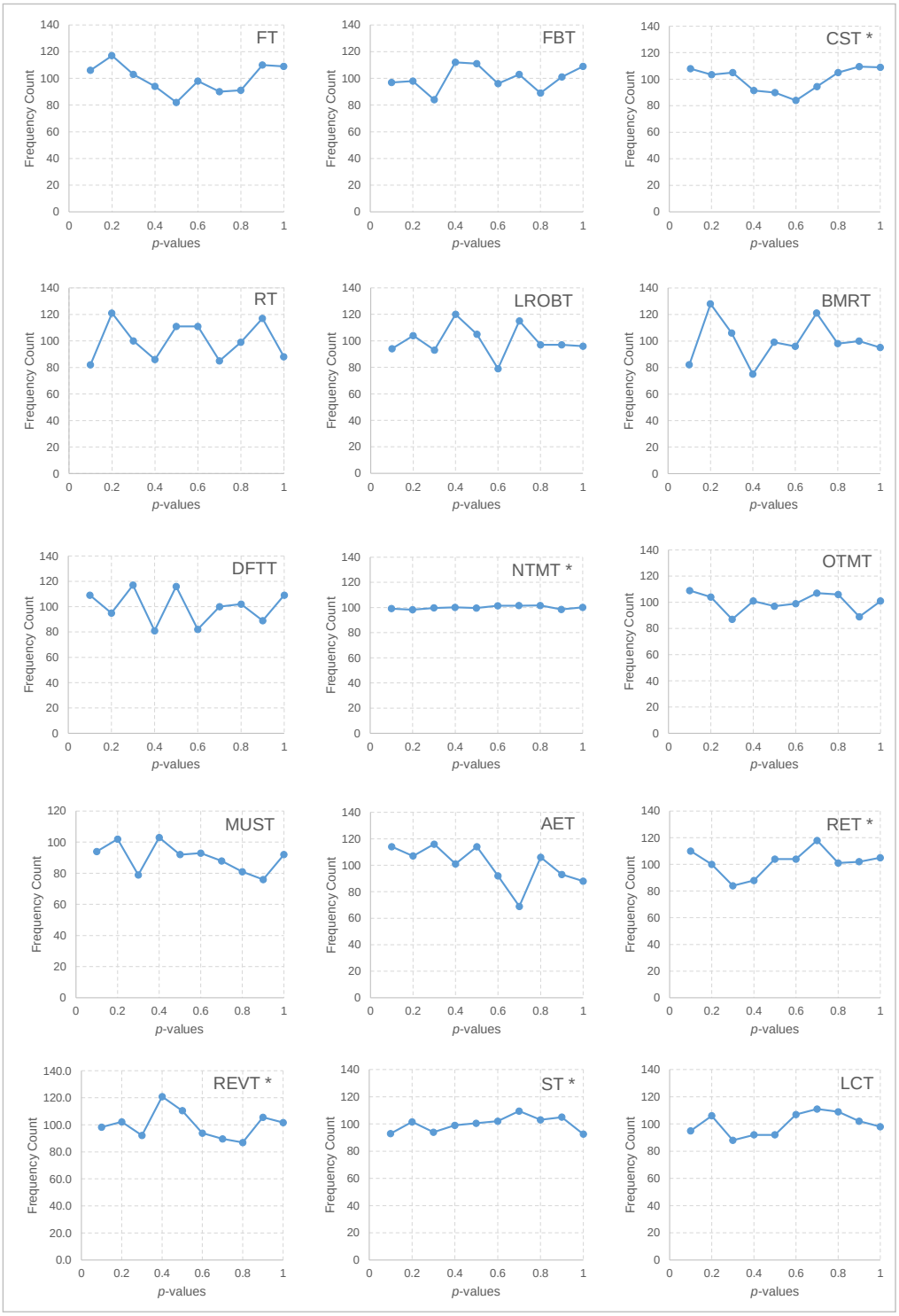


Figure 4·14: Uniform distribution of p-values. The tests with a * have their results in averaged values to save space.

Thus, the power consumption is mostly affected by the parameter N, as shown in Fig. 4·15.

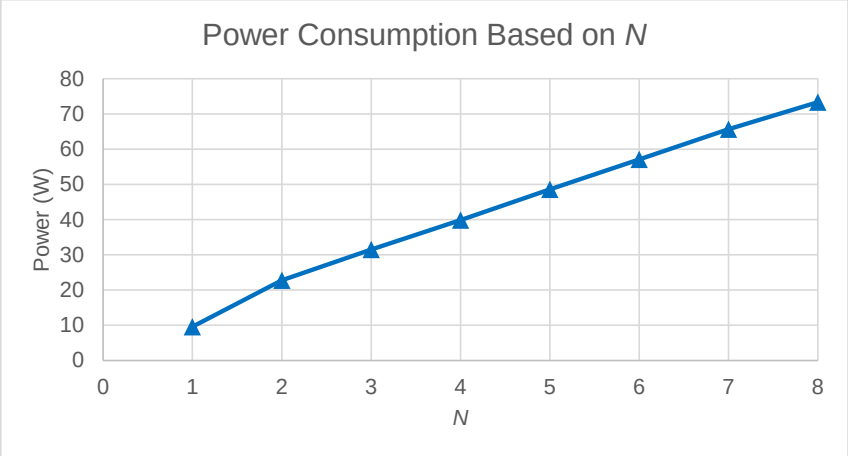


Figure 4·15: The power consumption is almost linearly proportional to the number (N) of active Lorenz functions.

Then, we denote "cost" as the energy (nJ) needed to produce one TRNG output bit. With the power data in Fig. 4·15, the throughputs listed in Section 4.5.1, and the p-values of all the possible configurations, we plot the cost-performance trade-off trend in Fig. 4·16.

Fig. 4·16 shows that the proposed design is able to work as a multi-purpose TRNG serving various demands of a system, as proposed in Section 4.2 and verified in Section 4.5.1. By tuning its parameters, the programmable TRNG can either function at a low cost with acceptable output quality, or achieve satisfying quality and security with more energy, or anything in between.

4.5.4 Output Entropy and Seed Sensitivity

In the proposed TRNG, there can be up to 128-bit true randomness seeds from the physical entropy source ASTRO. Obviously, this space can be easily expanded by using more ASTROs. While Fig. 4·8 shows that ASTRO alone already achieves a

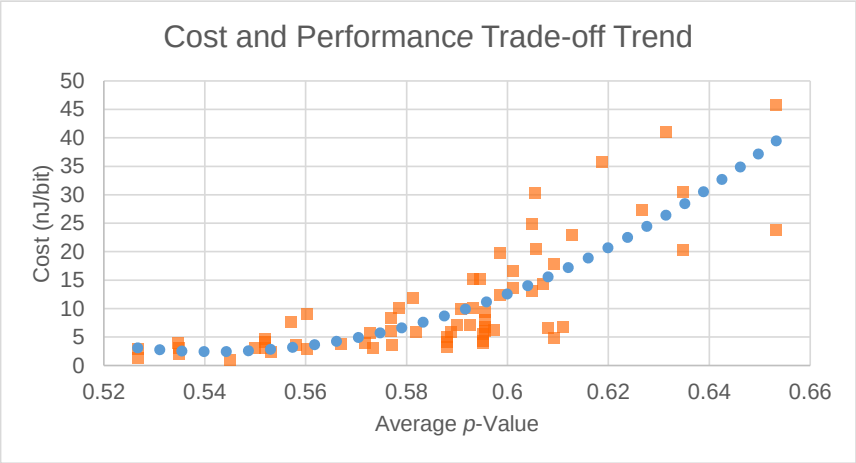


Figure 4·16: The blue dotted trend shows that the quality of the random sequences rises with the energy (nJ) consumed per bit.

relatively good entropy (0.964 bit/output bit), it can still be improved to a more satisfying degree, by the supporting modules in the TRNG architecture in Fig. 4·3.

Ideally, when the seed changes, the resulted random bit sequences should have as large Hamming Distance (HD) as possible. For the proposed TRNG three cases are studied:

1. Intra-TRNG with different seeds and user inputs;
2. Intra-TRNG with different seeds and same user inputs;
3. Inter-TRNG.

The following equation is used to calculate the sensitivity:

$$\text{Sensitivity} = \frac{2}{l(l-1)} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} \frac{HD(RBS_i, RBS_j)}{Len} \tag{4.5}$$

where $HD()$ is the Hamming Distance function, RBS_i the i^{th} random bit string, $Len = 2,000,000$ the length of each sequence, and l the number of sequences involved in evaluation. Table 4.3 shows the sensitivity of the three cases.

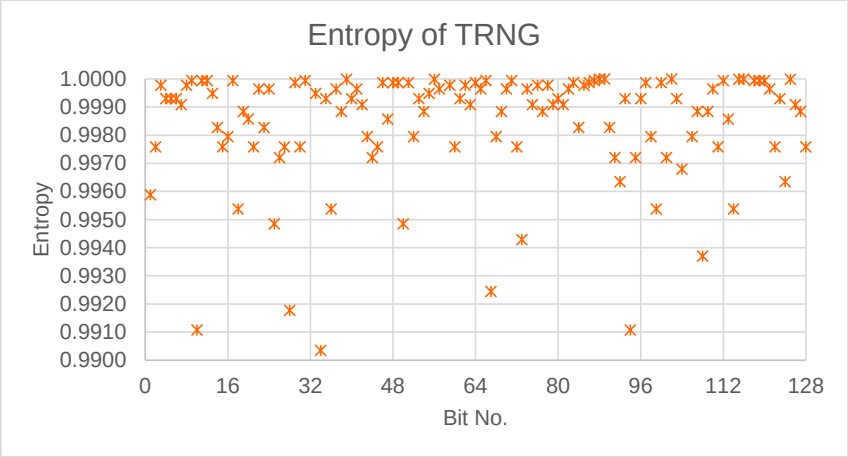


Figure 4·17: The entropy of the TRNG final outputs, which is 0.998 bit per output bit in average. To be comparable with Fig. 4·8, the TRNG is made to work under 128 bits/cycle throughput.

Table 4.3: Sensitivity Evaluation

	Intra-TRNG w/ different inputs	Intra-TRNG w/ the same input	Inter-TRNG
Sensitivity	50.0016%	50.0032%	49.9990%

[I] The sensitivity of the three cases are all around the ideal number 50%.

4.5.5 Comparisons with the Other Chaotic Map-based RNGs

As mentioned in Section 3.1, there have been quite a few works in the recent years to construct RNGs based on chaotic-maps. Lynnyk *et al.* (Lynnyk et al., 2015) (referred to as [1] in the following comparisons) proposed two PRNGs also based on Lorenz functions. Kim *et al.* (Kim et al., 2017) ([2]) proposed a low power TRNG based on 1-D linear piecewise affine Markov chaotic maps and analog-to-digital converter (ADC). François *et al.* (François et al., 2014) ([3]) tried to mix three logistic maps together. Dissipative quantum maps are used by Akhshani *et al.* (Akhshani et al., 2014) ([4]). Fraga *et al.* (de la Fraga et al., 2017) ([5]) studied different behaviors of RNGs based on three chaotic maps: Bernoulli shift, Tent, and Zigzag maps. Wang *et*

al. (Wang et al., 2016a) ([6]) built a PRNG based on an enhanced version of logistic maps. Özkaynak (Özkaynak, 2014) ([7]) used chaotic maps in a novel way as the additional input to an existing PRNG to improve its performance. However, they all lack flexible programmability for different applications.

In this subsection, the proposed TRNG is compared with the other chaotic map-based RNGs above on output quality, entropy, throughput, and energy cost. In each comparison, the proposed TRNG is configured to work on the corresponding mode respectively (cf. subsection 4.5.2).

First, in Fig. 4·18, a comparison on output quality (evaluated by NIST random test) is made between the proposed TRNG and these 7 competitors (referred to by their citation indexes).

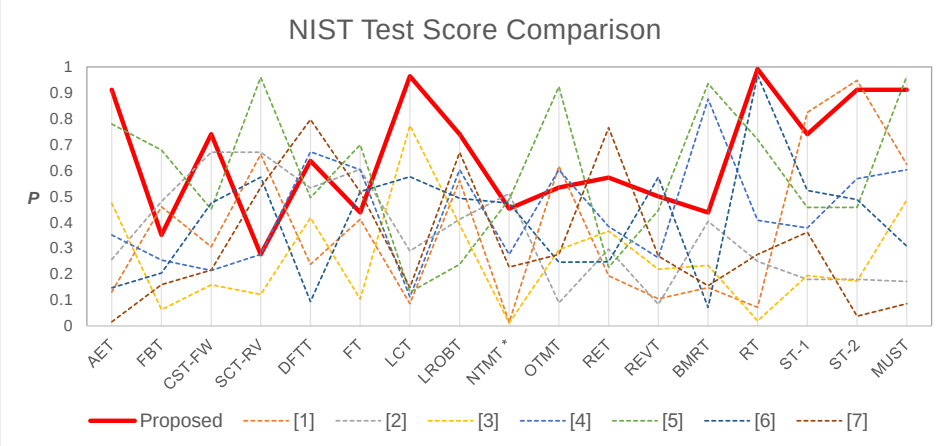


Figure 4·18: If a competitor has more than one design of RNG, the one with the best p-value is adopted in the figure.

By a closer look at the p-values in Table 4.4, when programmed to be working in the high quality mode (cf. Table 4.2), the proposed TRNG outscores the competitors. It has the best average p-value and more sub-tests with high p-values. It is also free of barely passing scores ($p \approx 0.01$) or even low scores ($p \approx 0.1$). In contrast, each competitor has at least one low score.

Then a comparison is made on entropy, throughput, and energy/bit. Thus the

Table 4.4: Statistical Comparison on p-values

	Proposed	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Average p	**0.65**	0.38	0.36	0.26	0.44	0.59	0.41	0.32
$p \geq 0.3$	**16**	9	8	6	11	14	11	6
$p \geq 0.5$	**11**	6	5	1	7	8	6	5
$p \geq 0.7$	**8**	2	0	1	1	6	1	2
$p \geq 0.9$	**5**	1	0	0	0	4	1	0
Low Scores	**0**	4	2	5	1	1	2	3
Barely Pass	**0**	1	0	2	0	0	0	1

[I] Most competitors have a large distance to the proposed TRNG in terms of the average p-value. The closest one is the Bernoulli shift map-based RNG in (de la Fraga et al., 2017) [5], and then the quantumn map-based RNG in (Akhshani et al., 2014) [4].

proposed RNG will work on its high-throughput and low-energy modes respectively (the entropy/bit remains the same for all configurations). Please note that some of the works on chaotic map-based RNGs did not provide data on all the aspects.

4.5.6 Comparisons with the Other Types of RNGs

Besides the chaotic map-based RNGs, we also select another seven representative RNGs of other types (Kohlbrenner and Gaj, 2004) (referred to as [8] in the following comparison), (Tsoi et al., 2003) ([9]), (Danger et al., 2009) ([10]), (Kwok and Lam, 2006) ([11]), (Wieczorek and Golofit, 2014) ([12]), (Martin et al., 2018) ([13]), (Cret et al., 2008) ([14]) for comparison. These RNGs work under various mechanisms such as dual-metastability-based, RO-based, hash-based, and open-loop-based etc. Most of them have high citation numbers and sufficient data for comparison.

Similar to Fig. 4·18, Fig. 4·19 shows the NIST random test scores by the p-values of each sub-test. The other RNGs are referred to by their citation indexes.

Table 4.6 gives a closer look at the p-values. Similar to the comparison in subsection 4.5.5, the proposed TRNG has the best average p-value and more sub-tests with

Table 4.5: Comparisons on Entropy, Throughput, and Energy/bit

	Entropy/bit (Physical)	Entropy/bit (RNG Output)	Throughput (Mbps)	Energy/bit (nJ/bit)
Proposed	**0.94**	**0.99**	**192**	**4.68**
[1]	N/A	N/A	N/A	N/A
[2]	N/A	0.99	0.27	0.0003
[3]	N/A	0.99	N/A	N/A
[4]	N/A	0.99	0.54	N/A
[5]	N/A	0.99	7.4	16.2
[6]	N/A	N/A	0.006	177.3
[7]	N/A	0.69	N/A	N/A

[I] It can be seen that in terms of entropy, throughput, and energy/bit, that the proposed RNG still outperforms its competitors.

[II] It is notable that (Kim et al., 2017) [2] achieves very low energy consumption because it is made by analog circuit. When only the FPGA based designs are considered, the proposed TRNG is the most energy-saving design.

high p-values. It is also free of barely passing scores ($p \approx 0.01$) or even low scores ($p \approx 0.1$). In contrast, each competitor has at least one low score.

4.5.7 Comparisons with a Combination of Multiple Traditionally Optimized RNGs

Among all the competitors, we select three RNGs which are traditionally optimized on high-quality and entropy ((de la Fraga et al., 2017)), high throughput ((Danger et al., 2009)), and low energy/bit ((Wieczorek and Golofit, 2014)) respectively. A comparison is made between the proposed work and the combination of the three.

The purpose of this comparison is to verify which is better: to have one programmable multi-purpose TRNG (the proposed work), or to have a combination of three traditional optimized RNGs?

Table 4.7 shows that the proposed programmable TRNG is able to outperform the RNGs which were optimized specifically for randomness quality and throughput.

Table 4.6: Statistical Comparison on p-values

	Proposed	[8]	[9]	[10]	[11]	[12]	[13]	[14]
Average p	**0.65**	0.50	0.31	0.40	0.34	0.22	0.52	0.32
$p \geq 0.3$	**16**	10	9	9	8	7	12	9
$p \geq 0.5$	**11**	8	4	8	5	1	9	4
$p \geq 0.7$	**8**	6	3	6	3	1	5	3
$p \geq 0.9$	**5**	3	1	0	2	1	2	1
Low Scores	**0**	2	7	7	7	10	1	6
Barely Pass	**0**	2	2	4	5	7	0	3

[I] Most competitors have a large distance to the proposed TRNG in terms of the average p-value. The closest one is the RO-based RNG under ionizing radiation in (Martin et al., 2018) [13].

Table 4.7: Comparisons with Conventionally Optimized RNGs

	Quality (Average p-value)	Entropy/bit (RNG Output)	Throughput (Mbps)	Energy/bit (nJ/bit)
Proposed	**0.65**	**0.99**	**192**	**4.68**
[5]: Quality	<u>0.59</u>	<u>0.99</u>	7.4	16.2
[10]: Throughput	0.40	0.99	<u>100</u>	N/A
[12]: Energy	0.22	0.98	30	<u>3.10</u>

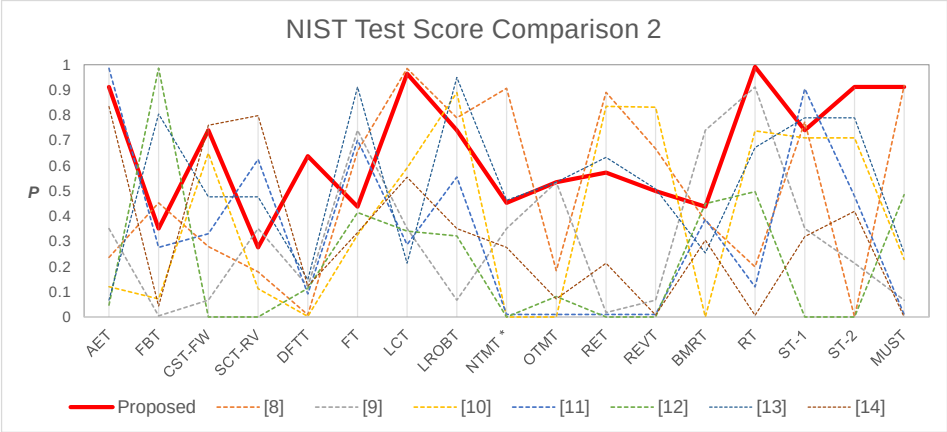


Figure 4·19: If a competitor has more than one design of RNG, the one with the best p-value is adopted in the figure.

Its energy consumption per bit is also very close to the RNG optimized for low power.

In one word, having one such programmable TRNG tends to be a better choice (in terms of both performance and cost) than having a combination of multiple traditional TRNGs, which are optimized for different purposes.

4.5.8 Comparisons with Other Ring Oscillators-based TRNGs

Since the proposed TRNG uses ring oscillators (RO) as its physical entropy source in ASTRO, we also compare it with other RO-based TRNGs. The most-cited RO-based RNG works of Sunar (Sunar et al., 2007) (referred to as [15]) and Wold (Wang et al., 2009) [16] are involved. Then some other works ((Tsoi et al., 2003) [17], (Martin et al., 2018) [18], (Cret et al., 2008)[19]) with sufficient details on entropy, throughput, and energy are also compared.

From Table 4.8, it can be seen that the proposed RNG performs better than other RO-based RNGs in terms of NIST scores, throughput, and energy cost per bit.

Table 4.8: Comparisons with Other RO-based TRNGs

	Quality (Average p-value)	Entropy/bit (RNG Output)	Throughput (Mbps)	Energy/bit (nJ/bit)
Proposed	**0.65**	**0.99**	**192**	**4.68**
[15] [16]	N/A	0.99	100	N/A
[17]	0.31	N/A	0.07	121.3
[18]	0.52	0.93	25	N/A
[19]	0.32	N/A	9.4	N/A

[I] (Wang et al., 2009)[16] is an improvement of (Sunar et al., 2007)[15] therefore we group their data together for this comparison.

4.6 Conclusion

In this chapter we propose a design of a programmable multi-purpose true random number generator (TRNG) based on Lorenz chaotic systems. The original purpose was to have it serve the cryptosystems in the proposed architectural model. However, as the research advanced, we found that high quality TRNG design is non-trivial and it is understood nowadays much less than it should be. Therefore, it extended to a larger project as we have seen in this chapter.

Through six configurable parameters, the proposed TRNG is able to generate any desired version by tuning the cost-performance trade-off. Therefore, it is able to serve the systems consisting of multiple modules with different demands on random sequences, e.g., budget-limited, high throughput, hight quality and security etc. More work can done to further extend this design, such as: 1) plugging in alternative chaotic maps, 2) using more efficient multipliers, and 3) designing better blender (BLD) formulas.

Chapter 5

Quantum-resistant Extension of Hardware Primitives

In order to construct the secure protocols in Chapter 2 (e.g., *island join* protocol 2.4.1, invisible *join* protocol 2.5.3, or group anonymous authentication protocol 2.5.4, etc.), we need two hardware primitives: Public-Key Cryptosystem (PKC) and Oblivious Transfer (OT).

As there are many out-of-box hardware implementations available for RSA, Elliptic Curve Cryptography or ElGamal, one option was to use one of these existing schemes as is. However, we take the approach to design and implement a set of quantum-resistant hardware primitives. The advantages of this approach include but not limited to: lower power consumption and smaller hardware area cost comparing with the classical cryptosystems, and a better chance to withstand the computing power of quantum computers.

Although there are different opinions on the feasibility of large scale quantum computers (Dyakonov, 2018), many estimations projected a time line of 5 to 10 years for universal quantum computers to be readily available. Particularly, in the last three years, we have witnessed a raft of breakthroughs and several key milestones towards the development of quantum computers. These rapid advances do bring with them critical challenges to classical cryptosystems. Most classical cryptosystems build their security reduction on integer factorization and discrete logarithm problems. However, there exist quantum computer-based algorithms to solve these

problems with relatively high efficiency, given universal quantum computers with a large number of logical qubits. For example, Shor's algorithm (Shor, 1999) leveraging quantum Fourier transform is able to solve the integer factorization problem by efficiently calculating the Carmichael's totient function.

Researchers have been actively investigating new algorithms and designs for cryptosystems in post-quantum era. In 2017, NIST launched a campaign of post-quantum cryptography standardization and totally 69 candidates were submitted. On January 30, 2019, 27 candidates made to the second round (semi-final) of this contest (NIST, 2019). Among all these submissions, lattice-based (12 candidates) and code-based (8 candidates) cryptosystems are the leading candidates. Particularly, designs based on Ring-learning with errors (R-LWE) (Lyubashevsky et al., 2010), which is an approximation of lattice-based cryptography, thus far have proven to be the most promising approach.

In this chapter we mainly focus on the hardware primitives based on R-LWE (Bu et al., 2019).

5.1 Quantum-resistant Hardware Primitives Based on Ring-Learning with Errors

Ring-Learning with Errors (R-LWE) was originally proposed by (Regev, 2009) as an approximation of lattice-based cryptosystem by simplifying the computations over polynomial rings. The ring parameters are carefully chosen to further reduce the computation complexity. The R-LWE-based cryptosystem has become the most promising candidate in NIST's post-quantum standard cryptography contest in all the three categories: public-key, key exchange, and digital signature. In contrast, code-based systems do not apply to digital signature schemes, and multivariate-based systems do not to public-key and key exchange.

R-LWE-based cryptosystems have the following advantages (i) their security reduction is a modification of the shortest vector problem (SVP), which are known to be NP-hard, and so far there are no efficient classical or quantum algorithms to solve them; (ii) they can support homomorphic encryption (HE) schemes; (iii) they have much smaller key size comparing with other cryptosystems; (iv) finally, they own the potential to more efficient hardware implementations than their classical competitors (e.g., RSA, ElGamal etc.).

In contrast to the extensive literature on the study and software implementation (mostly C++) of the Ring-LWE algorithm, there has been little work on its efficient hardware implementation. Moreover, out of the existing hardware implementations, very few of them focus on scalability and efficiency. One technical reason is that large finite field operations (thousands of bits for each vector) - which form the core computational kernel of R-LWE - remain a key challenge for many hardware designers.

Recently, a handful of works have explored the FPGA implementation of the Key Exchange (KEX) (Oder and Güneysu, 2017), and even less the Public-Key Cryptosystem (PKC) (Roy et al., 2014). There is also a general lack of discussion on the design and hardware implementation of other cryptographic primitives such as oblivious transfer (OT) and zero-knowledge proof (ZKP), which play critical roles in many applications, such as private machine learning and crypto-currencies using blockchain. Efficient and secure hardware implementation of these schemes remains a young research field.

Therefore, in this work, we construct a small representative set of reusable, standalone hardware modules of these quantum-resistant cryptographic primitives: PKC, KEX, OT, and ZKP. The major contributions of these hardware primitives are:

- *Algorithm:* novel proposals of the quantum-resistant OT and ZKP algorithms;
- *Parameterization:* a parameterizable design to generate variable sized primitives

to enable their deployment in small devices such as Internet of Things (IoT), and large computing platforms such as homomorphic encryption engines;

- *Implementation:* FPGA-tailored optimization to reduce the area and power cost.

These primitives can serve as the fundamental building blocks to aid hardware designers in constructing various quantum-resistant secure systems in the post-quantum era.

5.1.1 Primitive Algorithms

The detailed algorithms of the public-key cryptosystem (PKC) and key exchange (KEX) can be found in (Lyubashevsky et al., 2010) and (Alkim et al., 2016), respectively. To avoid redundancy, we will only briefly introduce them in Algorithms 7 and 10.

The oblivious transfer (OT) is an important primitive aiding the invisible *join* protocol 2.5.3 and group anonymous authentication protocol 2.5.4, where a 1-out-of-N OT scheme is used. This OT mechanism enables a receiver to choose and receive a certain piece of information out of many pieces from the sender, while having no knowledge to the other pieces. The sender also remains oblivious to the receiver's selection. The OT is a widely used protocol in privacy-preserving applications such as DNA database query (Katz and Malka, 2010) and private machine learning (Liu et al., 2017).

There have been a handful proposals of quantum-resistant OT (David et al., 2014) (Brakerski and Döttling, 2018). However, some of them are 1-out-of-2 OT (it can be used to build 1-out-of-N OT with a hierarchical construction though), while others incur high computation or communication complexity, which makes them unsuitable for resource-bounded hardware designs.

Algorithm 7: R-LWE Public Key Cryptosystem

Setup: Let the ring R_q be $R_q = R/\langle q \rangle = \mathbb{Z}_q[x]/\langle f(x) \rangle$, where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number. Thus R_q is a ring of integer polynomials modulo both $f(x)$ and q, and it has q^n elements. Let $\mathcal{X}$ be a discrete Gaussian distribution of small errors/noise centered around zero with standard deviation αp, where $\alpha < \sqrt{\log n/n}$. If $t = \lfloor \frac{q}{2} \rfloor$, $a, b \in R_q$ and $s, e, r_0, r_1, r_2 \leftarrow \mathcal{X}$, then the public key encryption protocol between Alice and Bob is as follows.

1 **Key Generation:** Alice picks a random $a \in R_q$ and samples $s, e \leftarrow \mathcal{X}$ to generate the public key $pk = \{a, b\}$ and the private key $sk = \{s\}$ by:

$$b = a \cdot s + e \tag{5.1}$$

where $\cdot$ is polynomial multiplication over the ring.

2 **Encryption:** Bob samples $r_0, r_1, r_2 \leftarrow \mathcal{X}$. He then converts his message into a binary vector (plaintext) m of length n, and generates the cipher $\{c_0, c_1\}$ as:

$$\begin{cases} c_0 &= b \cdot r_0 + r_2 + tm, \\ c_1 &= a \cdot r_0 + r_1. \end{cases} \tag{5.2}$$

3 **Decryption:** Alice decrypts the cipher by:

$$m = \lceil (c_0 - c_1 \cdot s)/t \rfloor \tag{5.3}$$

where $\lceil \quad \rfloor$ stands for taking the nearest binary integer.

We propose a simple universally composable 1-out-of-N OT (Bu et al., 2019). It is constructed on the foundation of the PKC primitive.

Algorithm 8: Oblivious Transfer Based on R-LWE Public Key Encryption

Setup: Let the ring R_q be $R_q = R/\langle q \rangle = \mathbb{Z}_q[x]/\langle f(x) \rangle$, where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number. Thus R_q is a ring of integer polynomials modulo both $f(x)$ and q, and it has q^n elements.

Let KeyGen() be the key generation function of the sender Alice, Enc() the encryption function of the receiver Bob, and Dec() the decryption function of Alice as in Algorithm 7. Alice has N n-bit binary messages $\{m_1, \cdots, m_N\}$ that Bob can choose from, and N n-byte random vectors $\{r_1, \cdots, r_N\}$ where $r_i \in R_q$. Then the oblivious transfer between Alice the sender and Bob the receiver is as follows.

Alice sends $\{r_1, \cdots, r_N\}$ to Bob. Bob chooses the c^{th} vector r_c in order to acquire m_c. Then Bob generates a random binary vector $K \in R_2^n$ and computes v to send to Alice:

$$v = r_c + \text{Enc}_{pk}(K) \tag{5.4}$$

where r_c is added to both ciphertexts $\{c_0, c_1\}$ (ref. [Eq.5.2])

1 For all $i \in \{1, 2, \cdots, N\}$, Alice computes the set $\{m_i'\}$ and sends it back to Bob:

$$m_i' = \text{Dec}_{sk}(v - r_i) \oplus m_i \tag{5.5}$$

where r_i is subtracted from both ciphertexts $\{c_0, c_1\}$. $\oplus$ is bitwise XOR.

2 Bob computes his desired m_c while remaining oblivious to other m_i if $i \neq c$:

$$m_c = m_c' \oplus K. \tag{5.6}$$

Remark 5.1.1. It is obvious that in order to breach the obliviousness, either Alice has to know K, or Bob has to know all the $\{r_1, \cdots, r_N\}$ and be able to break Dec(), namely acquiring the secret key by solving the SVP. □

In addition to PKC and OT, which are key building blocks for the secure protocols in Chapter 2, we also propose two more primitives: zero-knowledge proof (ZKP) and key exchange (KEX). The ZKP enables an entity to prove to a verifier that it knows a secret value s, without revealing any information apart from the fact that it knows the value.

Similarly to OT, there have been only a handful proposals of quantum-proof ZKP (Goldfeder et al., 2016) (Chase et al., 2017). However, these implementations too incur a large communication overhead (e.g., multi-party computation (MPC)

protocol). Hence, they are also impractical for resource-bounded hardware designs.

We propose a simple ZKP scheme (Bu et al., 2019) in Algorithm 9. It takes two rounds of interactions between Alice and Bob for the proof.

Algorithm 9: Zero-Knowledge Proof Based on R-LWE

Setup: Let the ring R_q be $R_q = R/\langle q\rangle = \mathbb{Z}_q[x]/\langle f(x)\rangle$, where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number. Thus R_q is a ring of integer polynomials modulo both $f(x)$ and q, and it has q^n elements. Let $\mathcal{X}$ be a discrete Gaussian distribution of small errors/noise centered around zero with standard deviation αp, where $\alpha < \sqrt{\log n/n}$. Let $t = \lfloor \frac{q}{2} \rfloor$, $a, b, s \in R_q$ and $e, r, e', u \leftarrow \mathcal{X}$. Suppose Alice has a secret s and needs to prove her ownership of it to Bob. It is notable that unlike the PKC scheme where $s \leftarrow \mathcal{X}$, in this ZKP scheme, s can be any arbitrary polynomial in R_q.

1 Alice picks a random $a \in R_q$ and samples $e, e', r \leftarrow \mathcal{X}$. Alice also selects an arbitrary binary vector m to compute:

$$\begin{cases} b &= a \cdot s + e, \\ c &= a \cdot r + mt + e' \end{cases} \tag{5.7}$$

where $\cdot$ is polynomial multiplication over the ring.

2 Alice sends $\{a, b, c, m\}$ to Bob without revealing s.

3 Bob samples $u \leftarrow \mathcal{X}$, and interactively sends it to Alice.

4 Alice responds with x to Bob:

$$x = r + s \cdot u. \tag{5.8}$$

5 Bob verifies if:

$$\lceil (c - a \cdot x + b \cdot u)/t \rfloor \stackrel{?}{=} m, \tag{5.9}$$

where $\lceil \quad \rfloor$ stands for taking the nearest binary integer.

6 If the equality of [Eq. 5.9] stands, then Alice has successfully proven her ownership of s to Bob.

Remark 5.1.2. It is obvious that given $\{a, b, c, m\}$, solving s for Bob is equivalent to solving the SVP in lattice-based cryptography, as proven by (Regev, 2009). However, by expanding [Eq. 5.9] we have:

$$\begin{aligned} &\lceil (c - a \cdot x + b \cdot u)/t \rfloor \\ &= \lceil (a \cdot r + \bar{m}t + e') - (a \cdot r + a \cdot s \cdot u) + (a \cdot s \cdot u + e \cdot u)/t \rfloor \\ &= \lceil (\bar{m}t + e' + e \cdot u)/t \rfloor = \lceil (\bar{m}t + \text{“small”})/t \rfloor \\ &= \bar{m} \end{aligned} \tag{5.10}$$

If $\bar{m} = m$, Alice proves her ownership of s without revealing it. □

Finally in Algorithm 10, we introduce the key exchange (KEX) scheme, also known

as the "NewHope" as proposed by (Lyubashevsky et al., 2010).

Algorithm 10: R-LWE Key Exchange

Setup: Let the ring R_q be $R_q = R/\langle q \rangle = \mathbb{Z}_q[x]/\langle f(x) \rangle$, where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number. Thus R_q is a ring of integer polynomials modulo both $f(x)$ and q, and it has q^n elements. Let $\mathcal{X}$ be a discrete Gaussian distribution of small errors/noise centered around zero with standard deviation αp, where $\alpha < \sqrt{\log n/n}$. Let Rec() and HelpRec() be reconciliation functions to eliminate the deviation caused by small errors (Alkim et al., 2016), and SHA3-256 a standard secure hash function. If $a, b, u, v \in R_q$ and $s, s', e, e', e'' \leftarrow \mathcal{X}$, then the key exchange protocol between Alice and Bob is as follows.

1 Alice picks random s and e and sends $\{a, b\}$ to Bob:

$$b = a \cdot s + e \tag{5.11}$$

where $\cdot$ is polynomial multiplication over the ring.

2 Bob picks random s' and e', e'' and sends $\{u, r\}$ to Alice:

$$\begin{cases} u & = a \cdot s' + e', \\ v & = b \cdot s' + e'', \\ r & = \text{HelpRec}(v). \end{cases} \tag{5.12}$$

3 **Key Agreement:**
Alice computes the following and acquires the secret key μ:

$$\begin{cases} w & = \text{Rec}(u \cdot s, r), \\ \mu & = \text{SHA3-256}(w). \end{cases} \tag{5.13}$$

Similarly, Bob computes the following and acquires the secret key μ:

$$\begin{cases} w & = \text{Rec}(v, r), \\ \mu & = \text{SHA3-256}(w). \end{cases} \tag{5.14}$$

5.1.2 Efficient and Secure Hardware Implementation of the Primitives

From Algorithm 7 to 10, we find that they all share the following common operations:

1. **Polynomial Addition:** [Eq. 5.1, 5.2, 5.4, 5.7, 5.8, 5.9, 5.11, 5.12];
2. **Polynomial Subtraction:** [Eq. 5.3, 5.5, 5.9];
3. **Scalar Multiplication:** [Eq. 5.2, 5.7];

4. **Scalar Division to the Nearest Binary Integer:** [Eq. 5.3, 5.9];

5. **Polynomial Multiplication:** [Eq. 5.1, 5.2, 5.3, 5.7, 5.8, 5.11, 5.12, 5.13];

Note: All operations are modular.

Software implementations for these operators can be done in a relatively straightforward way. Since latency is usually the only concern in software implementation of these algorithms, many operations are carried out in minimum cycles. For example, to conduct an addition between two polynomials of length n, the n component-wise coefficient modular additions can be done either partially or fully parallel. Resource planning is rarely a concern for this case.

However, when the operators are implemented on a hardware cryptographic primitive or co-processor (whose speed is much faster than software on a general purpose processor), resource utilizations such as area and power consumption become the major concerns. For example, a 20-bit modular multiplier alone costs about 300 LUTs on FPGAs. It will be utterly unwise to perform n ($n \geq 128$ for most cases) parallel multiplications in parallel, which will lead to high resource utilization.

Therefore, we suggest the following optimized hardware-oriented implementation of those operators without compromising the security. The pseudo codes below are written in Verilog style.

We propose Algorithms 11 and 12 (polynomial addition and subtraction) with a conditional assignment to avoid the use of expensive modulo operations.

Similarly, Algorithm 13 (scalar multiplication) also leverages conditional assignment to avoid performing the actual multiplication operation, since the polynomial multiplicand is a binary vector.

We also manage to avoid using expensive divisions in the scalar division operation in Algorithm 14. We leverage the facts that (1) all the numbers are within the range of $[0, q-1]$, and (2) the divisor is $(q-1)/2$. First, we measure the absolute difference

Algorithm 11: Polynomial Addition Over R_q

Setup: Let $a = \{a_0, \cdots, a_{n-1}\}, b = \{b_0, \cdots, b_{n-1}\}, c = a + b \in R_q$ be two n-byte polynomials.
Let the counter register i be initialized as $i = 0$.

```
if i < n then
    c[i] = a[i] + b[i] − (a[i] + b[i] ≥ q?q : 0)
    i = i + 1;
end
else
    Return c
end
```

Algorithm 12: Polynomial Subtraction Over R_q

Setup: Let $a = \{a_0, \cdots, a_{n-1}\}, b = \{b_0, \cdots, b_{n-1}\}, c = a - b \in R_q$ be two n-byte polynomials.
Let the counter register i be initialized as $i = 0$.

```
if i < n then
    c[i] = a[i] − b[i] + (a[i] ≥ b[i]?0 : q)
    i = i + 1;
end
else
    Return c
end
```

Algorithm 13: Scalar Multiplication Over R_q

Setup: Let $t = \lfloor q \rfloor$ be a constant, $m = \{m_0, \cdots, m_i, \cdots m_{n-1}\}$ where $m_i \in \{0, 1\}$. Then $c = tm \in R_q$.
Let the counter register i be initialized as $i = 0$.

```
if i < n then
    c[i] = (m[i] == 0?0 : t)
    i = i + 1;
end
else
    Return c
end
```

between the divisor and dividend. Then, we compare the difference with half of the divisor $((q-1)/4)$. If the difference is smaller, then it returns 0, otherwise 1.

Algorithm 14: Scalar Division to the Nearest Integer Over R_q

Setup: Let $t = (q-1)/2$ and $t_{\text{half}} = t/2$ be constants. Also let $a = \{a_0, \cdots, a_i, \cdots a_{n-1}\} \in R_q$ and $c = \lceil a/t \rfloor \in R_2$.
Let the counter register i be initialized as $i = 0$.

if $i < n$ **then**
  $c[i] = ((a[i] \geq t)?(a[i] - t) : (t - a[i])) < t_{\text{half}}?1 : 0$
  $i = i + 1$
end
else
  Return c
end

The hardware micro-architectures realizing Algorithms 11 - 14 are shown in Fig. 5·1.

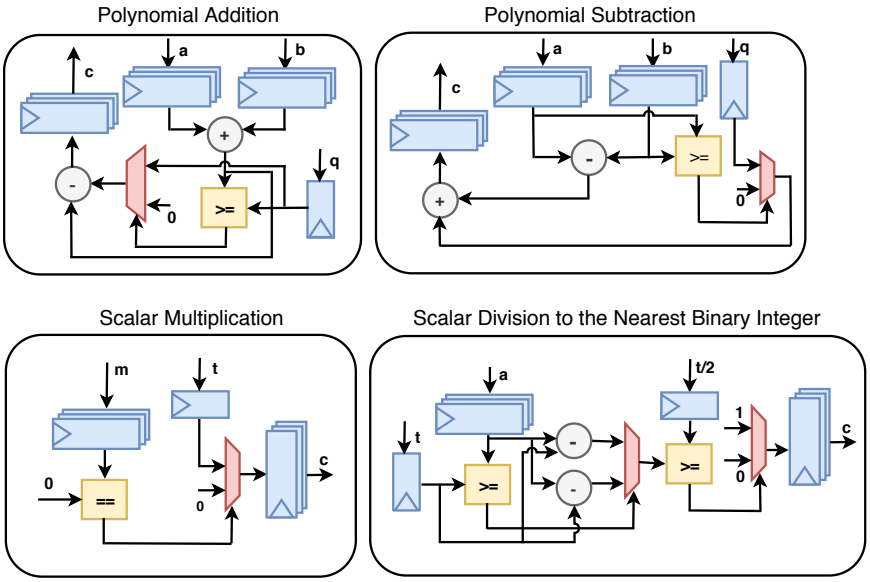


Figure 5·1: The four basic operations of the PKC algorithm: Polynomial Addition, Polynomial Subtraction, Scalar Multiplication, and Scalar Division to the Nearest Binary Integer. With careful design and optimization, we manage to perform all the four modulo operations over R_q without using a general mod q reduction module.

Polynomial Multiplication

All the four operations mentioned in the previous subsection are component-wise operations and can be implemented using conditional assignment. This approach reduces the hardware resource utilization significantly. However, the polynomial multiplication is essentially a convolution operation and has the highest hardware cost. An efficient multiplication module will substantially improve hardware cost of the entire crypto-primitive.

A straightforward convolution between two n-byte vectors requires $O(n^2)$ multiplications. However, by the convolution theorem, using fast discrete Fourier transform (DFT) will take the complexity down to $O(n \log n)$ multiplications.

For polynomial multiplication over the ring, the butterfly Number-Theoretic Transform (NTT) with Negative Wrapped Convolution (NWC) are used instead of DFT. The use of NWC eliminates both polynomial length expansion and irreducible polynomial reduction during multiplication. Hence, the combination of NTT and NWC will largely improve the efficiency of polynomial multiplications (Chen et al., 2015).

The multiplication algorithm using NTT is described in Algorithm 15.

The major contribution of this implementation of NTT-based polynomial multiplier, is a parameterizable NTT/iNTT algorithm fully tailored and optimized for hardware implementation. Although there exist a number of parameterizable FFT/DFT algorithms, they perform a large number of multiplications and divisions to compute the indexes of points and twiddle factors. While this may not be an issue for software implementation, it quickly adds up to the hardware cost in both area and power.

In the proposed algorithm, instead of multiplication and divisions, all index calculations performed are shift, XOR, and conditional assignments, which are mostly cost-efficient operations in hardware. It is notable that since the proposed NTT algorithm uses many bit level manipulations, this algorithm does not translate to software

Algorithm 15: NTT-based Polynomial Multiplier

Let $a = \{a_0, \cdots, a_{n-1}\}, b = \{b_0, \cdots, b_{n-1}\} \in \mathbb{Z}_q[x]/\langle f(x)\rangle$ be two polynomials of length n (with n coefficients), where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number).
Let ω be the n-th root of unity, ω^{-1} the inverse of ω, $\phi^2 = \omega \bmod q$, and ϕ^{-1} the inverse of ϕ.

Precompute: $\{w^i, w^{-i}, \phi^i, \phi^{-i}\}$ for all $i \in \{0, 1, \cdots, n-1\}$
/ negative wrap convolution (NWC) of a and b */*
for *i=0 to n-1* **do**
  $\bar{a}_i \leftarrow a_i \phi^i$
  $\bar{b}_i \leftarrow b_i \phi^i$
end
/ number-theoretic transforming (NTT) a and b */*
$\bar{A} \leftarrow NTT^n_\omega(\bar{a})$
$\bar{B} \leftarrow NTT^n_\omega(\bar{b})$
/ component-wise multiplying A and B */*
$\bar{C} = \bar{A} \cdot \bar{B}$
$\bar{c} \leftarrow iNTT^n_\omega(\bar{C})$
/ inverse negative wrap convolution (iNWC) of c */*
for *i=0 to n-1* **do**
  $c_i \leftarrow \bar{c}_i \phi^{-i}$
end
Return c

implementation easily.

Fig. 5·2 shows the corresponding multiplier and NTT hardware implementation. The proposed implementation, when compared with (Chen et al., 2015), is highly simplified as shown in the figure.

Figure 5·3 shows the fully parameterizable R-LWE PKC system architecture using the operators built in this section.

5.1.3 Cost and Performance

We implemented the PKC system on a Zynq-7000 FPGA. Synthesis of the implementation is carried out using Xilinx Vivado 2018.2 design suite.

We present the correlation between the system parameter $\{n, q\}$ and costs (latency and hardware) for all the three building blocks of PKC system. The equations in table 5.1 provide an estimation of the performance and upon the selected $\{n, q\}$. Therefore,

Algorithm 16: Number-Theoretic Transform Fully Optimized for Hardware Implementation

Let $a = \{a_0, \cdots, a_{n-1}\} \in \mathbb{Z}_q[x]/\langle f(x)\rangle$ where where n is a power of 2, and $q \equiv 1 \bmod 2n$ is a prime number. Let ω be the n-th root of unity for q. Let ω^{-1} be the inverse of ω. Precompute ω^i and ω^{-i} for $i \in \{0, 1, \cdots, n/2-1\}$, and store them in the array element $\omega[i]$ and $i\omega[i]$ respectively.

Let a be swapped to A so that $A[j] = a[j_{rev}]$, where j_{rev} is a binary vector bit-reversed from j. Let $i, Stage$ both be initialized to 0.

Index Computation:

/ calculate the correspding point's index i_{corr} to the i^{th} point */*

assign $i_{corr} = i$ XOR $(1 << Stage)$;

/ calculate the twiddle factor k for both i_{corr} and i */*

assign $k_0 = (Stage == 0)?0 : (i << (\log n - Stage))$;

assign $k = k_0[\log q - 1 : 1]$;

Shared Variable Computation:

assign $v = A[i_{corr}] * \omega[k] \bmod p$;

NTT Function:

```
if Stage < log n then
    if i < n then
        i = i + 1;
        if i == n − 1 then
            Stage = Stage + 1;
        end
        if i[Stage] == 0 then
            A[i] = A[i] + v − (A[i] + v ≥ q?q : 0);
            A[i_corr] = A[i] − v + (A[i] ≥ v?0 : q);
        end
    end
end
else
    Return A as the transformed polynomial of a.
end
```

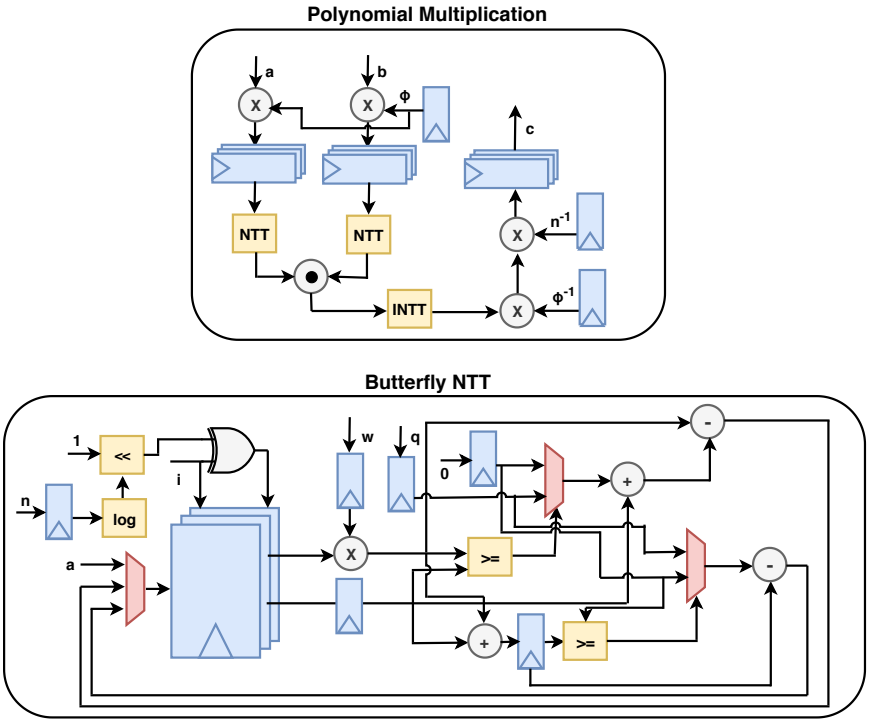


Figure 5·2: The schematic diagram for butterfly NTT operation.

it helps researchers to plan ahead on hardware resources and latency expectations in their designs.

As shown in table 5.1, the latency (in clock cycles) can be precisely computed as a function of n, and the hardware cost is proportional to $n \log_2 n \log_2 q$.

Table 5.2 presents the hardware implementation results for the entire PKC system. The hardware cost is proportional to the length of the polynomial and size (bit-width) of the chosen prime number. Synthesis results are generated using various polynomial length n, while the prime number q is set to 12,289.

We also present the hardware cost trend for different combinations of n and q in Figure 5·4. These trends further prove and confirm the correctness of the estimations given in Table 5.1.

In table 5.3, we present the latency of the PKC system with varying polynomial length n. We can see that the latency (in clock cycles) is determined by n and

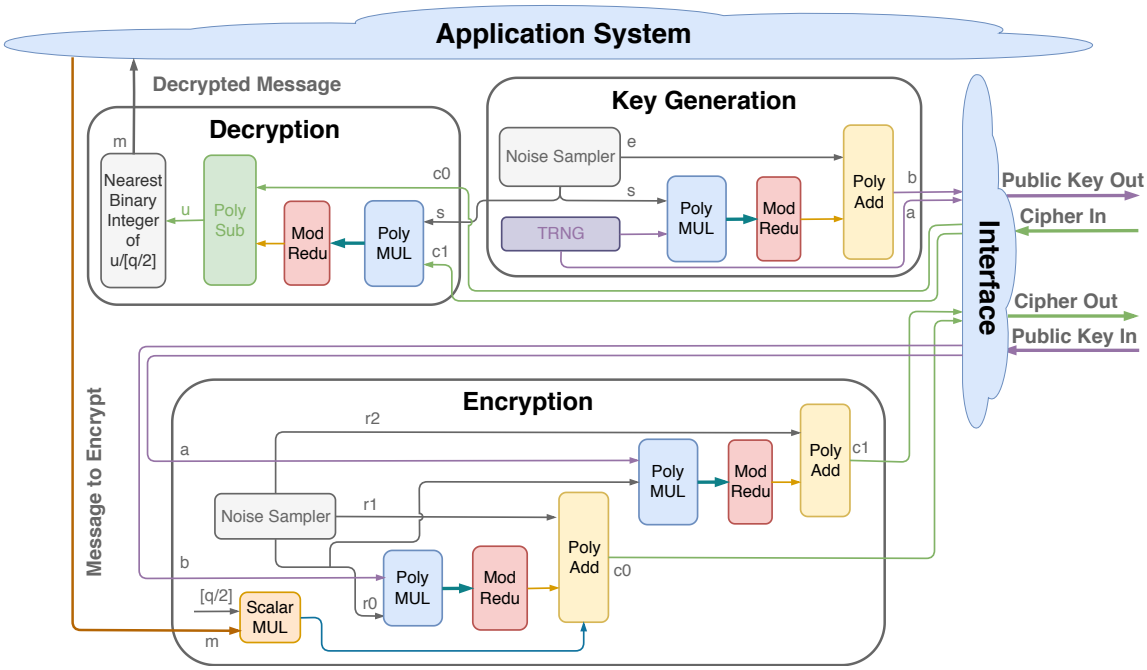


Figure 5·3: The three core building blocks for the primitives: *Key Generation (KeyGen)*, *Encryption (Enc)*, and *Decryption (Dec)*. Therefore, it can be used as either the public key distribution party, or the encryption party.

Table 5.1: Correlation between $\{q, n\}$ and Latency & Area for KeyGen, Enc and Dec Modules

Operation	Latency
KeyGen	$3n + \frac{3n}{2}\log_2 n$
Enc	$7n + 2n\log_2 n$
Dec	$4n + n\log_2 n$
Resource	**Cost**
LUTs	$O(n \log_2 n \log_2 q)$
Registers	$O(n \log_2 n \log_2 q)$

independent of q. However, it is notable that the maximum frequency will be reduced as q increases, due to the increasing combinational logic for the mod q operation.

Table 5.2: Hardware Cost with different n and q=12289

Length n	LUTs	Registers	DSP
8	9578	1104	26
16	12961	2157	26
32	19812	4273	26
64	31510	8340	26
128	66251	16805	26
256	11490	33138	26
512	227458	65643	26
1024	426402	130540	26

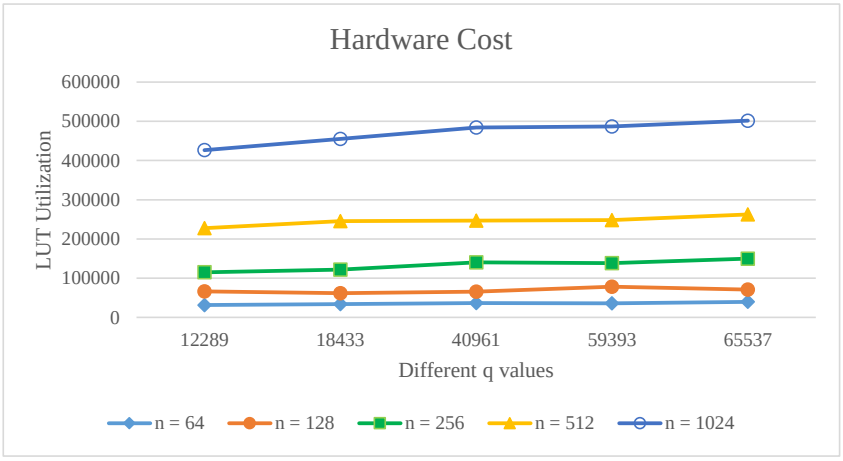


Figure 5·4: Hardware Cost with different q and n for PKC System

We also compare the latency of the proposed design implementation for NTT Multiplier to the design of (Chen et al., 2015) and (Pöppelmann and Güneysu, 2012) as shown in Figure 5·5. The figure shows the proposed implementation brings in a performance improvement of 23% comparing with Chen et al. (Chen et al., 2015) and 71% with (Pöppelmann and Güneysu, 2012).

Table 5.3: Latency (cycles) for KeyGen, Enc and Dec Modules in PKC

Length n	KeyGen	Enc	Dec
8	96	152	80
16	240	368	192
32	576	864	448
64	1344	1984	1024
128	3072	4480	2304
256	6912	9984	5120
512	15360	22016	11264
1024	33792	48128	24576

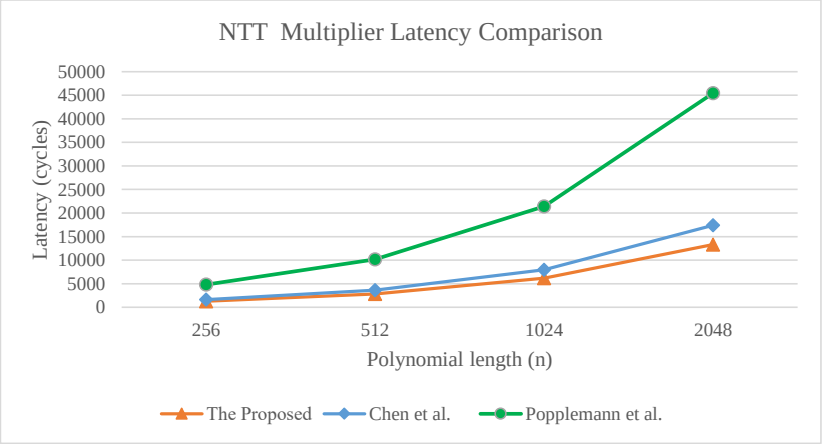


Figure 5·5: Latency comparison of the NTT-based Multiplier between the proposed design, and the designs of (Chen et al., 2015), and (Pöppelmann and Güneysu, 2012)

.

Chapter 6

System Performance Evaluation of the Proposed Architectural Model

The effectiveness of the protection provided by the the new architectural model to the attack models outlined in Chapter 2 is measured through flows and processes isolation. Strict isolation of flows and processes based on trust levels effectively creates control access to shared resources: (1) shared memory regions in the network, i.e., virtual channels at the router; and (2) distributed shared main memory modules. The degree of process isolation provided by the new architectural model inversely corresponds to the computing system attack surface, where higher isolation means smaller attack surface.

For the high-fidelity assessment of the system performance and hardware overheads, we adopt a register-transfer level (RTL) based evaluation approach in the experiments. We use the *Heracles* (Kinsy et al., 2013) (Kinsy et al., 2011), the *HAsim* (Pellauer et al., 2011), and the *BRISC-V* (Bandara et al., 2019) RTL design and simulation platforms for the development and evaluation of the proposed architectural model.

To illustrate and summarize the system performance evaluation, an 8×8 2-D mesh topology design is implemented on a Xilinx Virtex7-XC7VX690T FPGA device. The board has $433,200$ LUTs and $866,400$ register slices. The unmodified *switch allocation* step in the routing process has the critical path due to the arbitration scheme logic. The operating frequency is 151.5 MHz across all the designs. For the

power estimates, the Xilinx Power Estimator (XPE) in the Vivado Design Suites is used. The power numbers are the post-routing estimates using a vector based switching activity format (i.e., SAIF). The process feature was set to maximum, the airflow to 500 LFM and the power supply to default.

The router has four virtual channels and eight slots per virtual channel. *Heracles*' injector cores are used to create network and memory traffic. Table 6.1 shows the FPGA synthesis results. In the table, BA stands for baseline architecture—*Heracles*' seven-stage in-order RISC processor, AES the 128-bit version and KS stands for key storage unit, and PA the proposed architecture model. The hardware overhead to fully implement the security features of the new architectural model is only 17%. Table 6.2 has the power estimates for the different design. These estimates correlate fairly well to the logic resource utilization.

Table 6.1: FPGA implementation resource utilization.

Resource	**BA**	**BA + AES**	**BA + AES + KS**	**PA**
Regs	391,054	424,298	437,980	472,864
LUT	277,038	299,202	307,512	324,251
Percentage	-	8	11	17

Table 6.2: FPGA power estimates using Xilinx Power Estimator (XPE).

Power Estimates	**BA**	**BA + AES**	**BA + AES + KS**	**PA**
Dynamic	57.9	62.54	64.27	67.75
Device Static	6.45	6.96	7.16	7.54
Total On-Chip Power (W)	64.48	69.64	71.57	75.44

The trust level per core is randomly assigned. The synthetic benchmarks, *Uniform random*, *Bit complement* (BitComp), *Shuffle*, *Transpose*, and *Bit reverse* (BitRev),

are used for the various evaluations. Figure 6·1 shows the interactions among the different traffic classifications. For example, in the *Uniform random* benchmark, less than 2% of the highly secure traffic is interacting (i.e., sharing a physical link or buffer space or memory block) with other types of traffic. The limited interaction reduces the attack surface and ensures greater security.

Figures 6·3 and 6·2 show throughput and latency results of all the synthetic benchmarks on the baseline and the proposed architectural model. The graphs are color-coded to match the trust-level (e.g., HA BitRev green means that the *Bit reverse* traffic is run in the highly secure mode). Overall, the proposed secure architecture model shows no significant performance penalty. In some cases, we actually see a performance improvement at higher injection rates. This improvement occurs because the proposed architectural model tries to confine the traffic to their trust classification islands, which decreases path diversity and throughput for low injection rates but reduces *head-of-line-blocking* at high injection rates. In other words, by maximizing within-island routing, as a secondary effect, the algorithm also provides better traffic distribution and load balancing in the network.

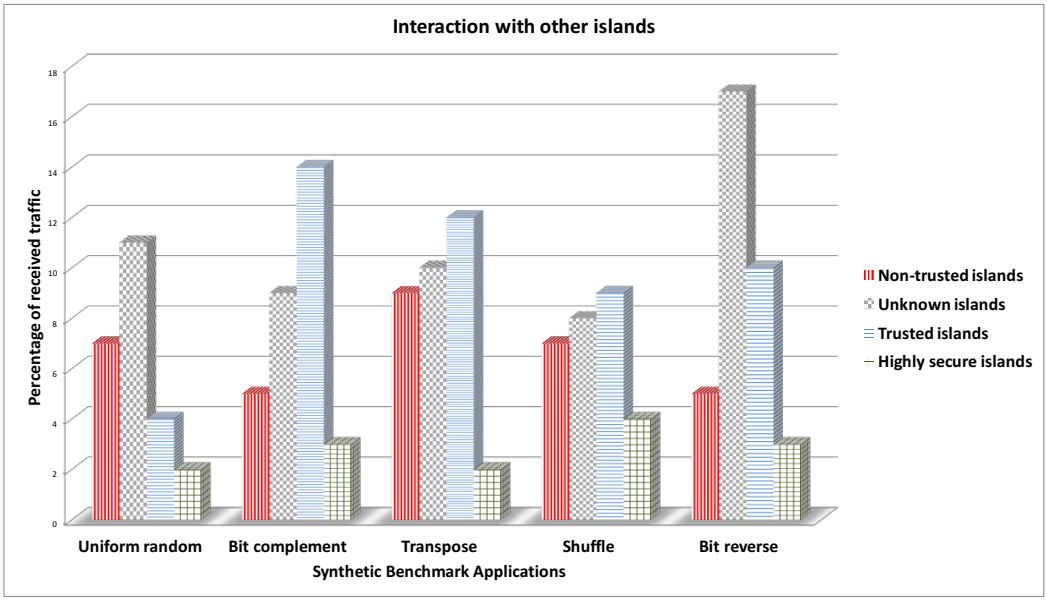


Figure 6·1: Percentage of interaction with other islands for different traffic classifications.

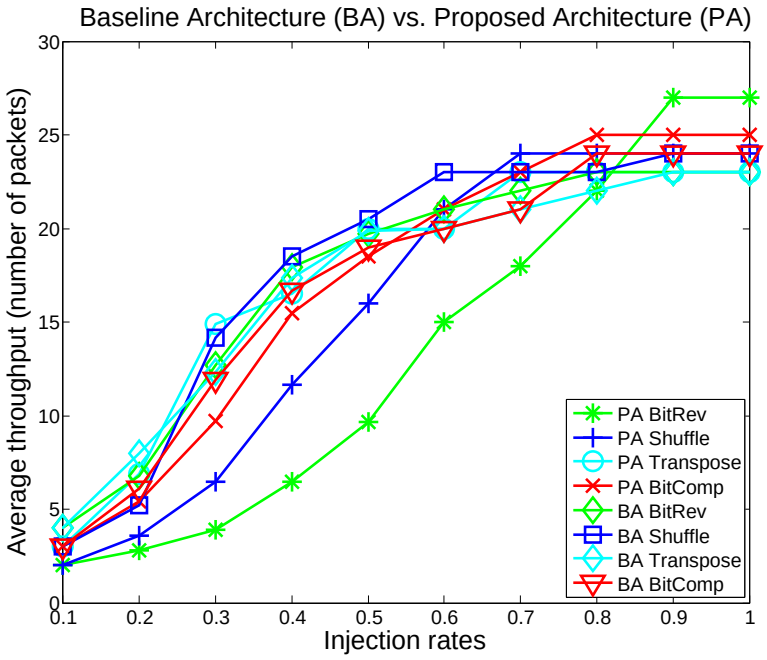


Figure 6·2: Throughput per benchmark for the baseline and the proposed architectural model.

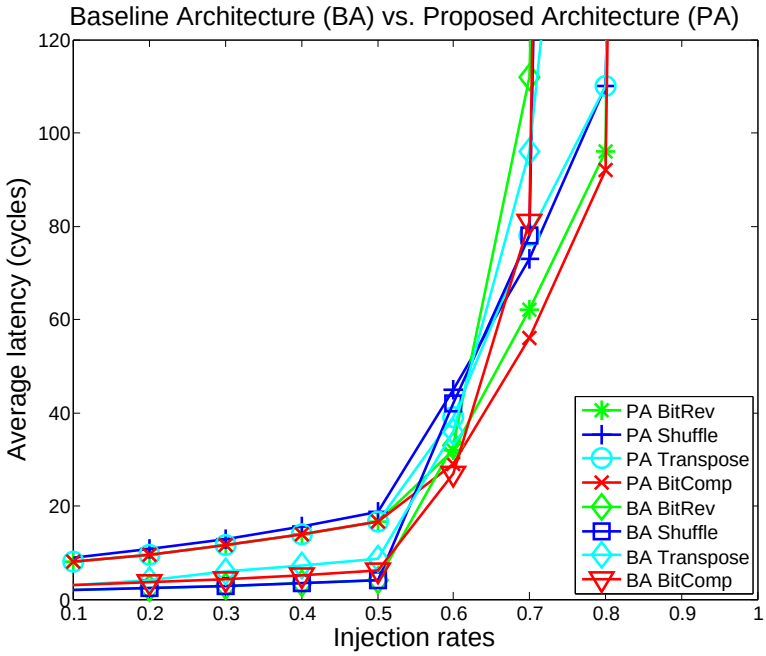


Figure 6·3: The average latency per benchmark for the baseline and the proposed architectural model.

In addition to the synthetic benchmarks, applications from the SPLASH-2 benchmark suite are used to evaluate the new architectural model's network performance and security enforcement. SPLASH-2 (Woo et al., 1995) is a diverse pool of applications commonly used to test and evaluate an architecture or a microarchitecture feature performance in a shared memory setting. These applications are simulated as traces. The Graphite (Miller et al., 2010) distributed x86 multicore simulator is used to generate the traces. Figure 6·4 shows the throughput results for the new architectural model on the SPLASH-2 benchmarks as fractions of throughput on the baseline architecture. The performance decline is only 1% to 9% across all the benchmarks when compared to the non-secure baseline architecture.

Figure 6·5 shows the degree of interaction among traffic for the different applications, measured as percentages. For these results, 25 placements (of processes and data) and simulations are performed and the best results are reported. *Non-*

interacting means that flows from the application are routed within the routers assigned to the application without needing to use other applications' island or local memory.

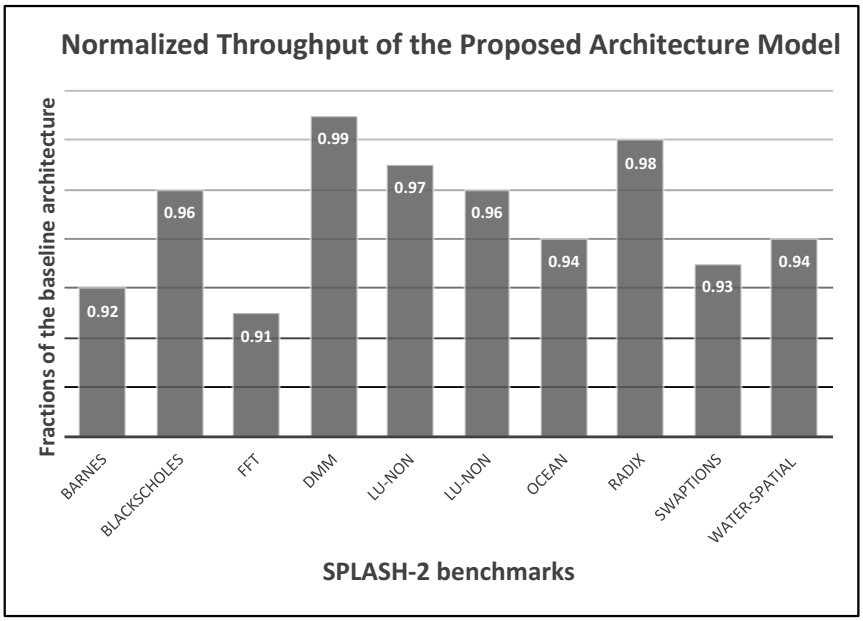


Figure 6·4: Throughput per benchmark in SPLASH-2 suite for the new architectural model.

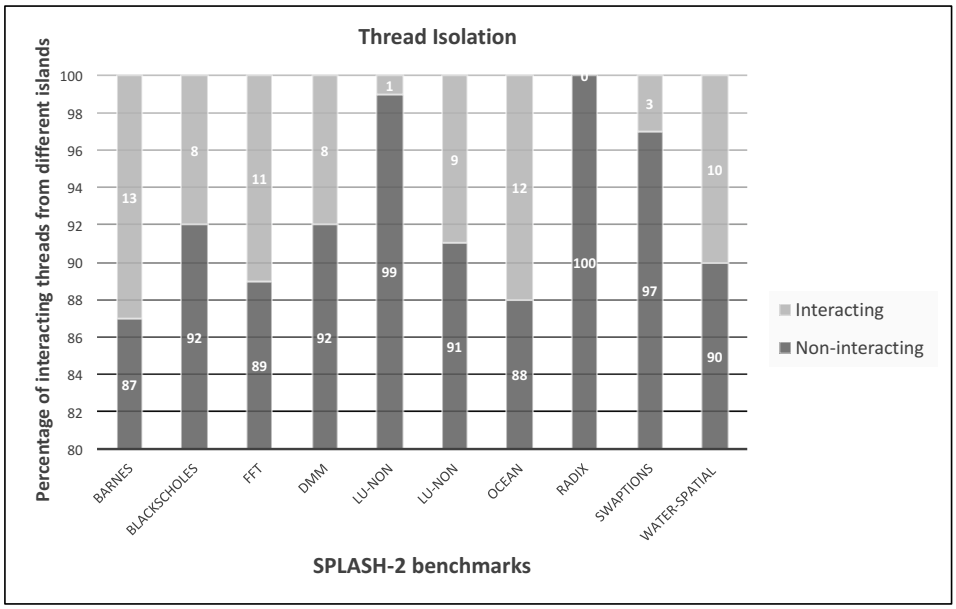


Figure 6·5: Degree of interaction among SPLASH-2 application traffic, measured as percentages.

Chapter 7

Conclusions

In this dissertation we propose a new architectural model aiming to build *secure* computer system out of *non-secure* and *untrusted* processing elements (PEs). It intends to address the security issues in heterogeneous SoCs caused by varying levels of security and trust of the PEs. Such vulnerability can lead to data leakage from the shared memory resources or denial of service attacks. To address these issues, the proposed architectural model takes 3 major approaches: (1) process isolation through hardware virtualization. Based on PEs' runtime trust levels, they are dynamically grouped to different virtual islands, which are logically isolated. Data leakage from trusted islands to less trusted ones are thus prohibited; (2) hardware-based root-of-trust, which provides a trustworthy channel to authenticate hardware components and resist counterfeits or spoofing; (3) a set of secure protocols to protect the system from untrusted PEs' malicious behaviors and to preserve the privacy of the trusted ones.

In order to extend the architectural model's usability to the post-quantum era, we use a set of quantum-resistant cryptographic hardware primitives as the basic building blocks for the secure protocols in this architectural model. These primitives can also aid designers to construct larger secure systems for the post-quantum era.

The proposed architectural model is evaluated by the Heracles RTL simulator with a set of benchmarks. The results show that the architectural model introduces acceptable hardware and throughput overheads while successfully carrying out the

protection to heterogeneous systems.

Our future work consists of the following parts: (1) to extend the capability of the proposed architectural model in detecting and resisting malicious components; (2) to reduce the hardware overhead of this architectural model through more efficient hardware implementations and algorithms; (3) to address the problem of elevating data from a less trusted island to a highly trusted with data sanitization; and (4), to construct the non-binary OLSC-based McEliece cryptosystem and verify its security level. We aim to explore the possibility of using OLSC-based McEliece cryptosystem as an approach to reduce decryption complexity and key size.

References

Aarestad, J. et al. (2013). Help a hardware-embedded delay puf. *IEEE Design and Test*.

Aharonov, D., Ben-Or, M., Eban, E., and Mahadev, U. (2017). Interactive proofs for quantum computations. *arXiv preprint arXiv:1704.04487*.

Akhshani, A., Akhavan, A., Mobaraki, A., Lim, S.-C., and Hassan, Z. (2014). Pseudo random number generator based on quantum chaotic map. *Communications in Nonlinear Science and Numerical Simulation*, 19(1).

Alkim, E., Ducas, L., Pöppelmann, T., and Schwabe, P. (2016). Post-quantum key exchange-a new hope. In *USENIX Security Symposium*, volume 2016.

Anderson, J. H. (2010). A puf design for secure fpga-based embedded systems. *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*.

Aron, J. (2019). Ibm unveils its first commercial quantum computer. https://www.newscientist.com/article/2189909-ibm-unveils-its-first-commercial-quantum-computer/.

Bandara, S., Ehret, A., Kava, D., and Kinsy, M. A. (2019). BRISC-V: an open-source architecture design space exploration toolbox. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, page 306.

Barker, E. and Roginsky, A. (2011). Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, 800:131A.

Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., Leigh, S. D., Levenson, M., Vangel, M., Banks, D. L., et al. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. *National Institute of Standards & Technology*.

Bauder, D. (1983). An anti-counterfeiting concept for currency systems. *Research report PTK-11990. Sandia National Labs*.

Berlekamp, E. (2015). *Algebraic coding theory: Revised Edition*. World Scientific.

Brakerski, Z. and Döttling, N. (2018). Two-message statistically sender-private ot from lwe. In *Theory of Cryptography Conference*, pages 370–390. Springer.

Bu, L., Agrawal, R., Cheng, H., and Kinsy, M. A. (2019). Post-quantum cryptographic hardware primitives. *Boston Area Computer Architecture Workshop, arXiv preprint arXiv:1903.03735*.

Bu, L., Cheng, H., and Kinsy, M. A. (2018a). Adaptive and dynamic device authentication based on lorenz chaotic systems. In *61st International Midwest Symposium on Circuits and Systems (MWSCAS)*.

Bu, L., Cheng, H., and Kinsy, M. A. (2018b). Fast dynamic device authentication based on lorenz chaotic systems. In *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE.

Bu, L., Isakov, M., and Kinsy, M. A. (2018c). A secure and robust scheme for sharing confidential information in iot systems. *Ad Hoc Networks*.

Bu, L. and Karpovsky, M. G. (2017). A design of secure and reliablewireless transmission channel for implantable medical devices. *3rd International Conference on Information Systems Security and Privacy*.

Bu, L. and Kinsy, M. (2018a). Weighted group decision making using multi-identity physical unclonable functions. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 251–2514. IEEE.

Bu, L. and Kinsy, M. A. (2018b). Weighted group decision making using multi-identity physical unclonable functions. In *28th International Conference on Field Programmable Logic and Applications (FPL)*.

Bu, L., Mark, M., and Kinsy, M. A. (2018d). A short survey at the intersection of reliability and security in processor architecture designs. *IEEE Computer Society Annual Symposium on VLSI*.

Bu, L., Nguyen, H. D., and Kinsy, M. A. (2017). Rasss: A perfidy-aware protocol for designing trustworthy distributed systems. In *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE.

Byrne, M. (2015). Internet of things encryption vulnerabilities show how often devs rip-off code. https://motherboard.vice.com/en_us/article/nz7v77/internet-of-things-encyption-vulnerabilities-show-how-often-people-rip-off-code.

Byrne, M. (2016). Is computer security becoming a hardware problem? https://motherboard.vice.com/en_us/article/z43qn9/is-computer-security-a-hardware-problem.

Camenisch, J., Hohenberger, S., and Lysyanskaya, A. (2005). Compact e-cash. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.

Cao, Z., Chen, F., Chen, B., and Zhang, X. (2015). Research on the balanced boolean functions satisfying strict avalanche criterion. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 680–684. IEEE.

Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., and Zaverucha, G. (2017). Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1825–1842. ACM.

Chatterjee, U., Govindan, V., Sadhukhan, R., Mukhopadhyay, D., Chakraborty, R. S., Mahata, D., and Prabhu, M. M. (2018). Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database. *IEEE Transactions on Dependable and Secure Computing*.

Chen, D. D., Mentens, N., Vercauteren, F., Roy, S. S., Cheung, R. C., Pao, D., and Verbauwhede, I. (2015). High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):157–166.

Chen, Q. A., Qian, Z., and Mao, Z. M. (2014). Peeking into your app without actually seeing it: Ui state inference and novel android attacks. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 1037–1052, Berkeley, CA, USA. USENIX Association.

Chhabra, S., Solihin, Y., Lal, R., and Hoekstra, M. (2010). An analysis of secure processor architectures. In *Transactions on computational science VII*, volume 5890, pages 101–121. Springer-Verlag.

Chu, C.-K. and Tzeng, W.-G. (2008). Efficient k-out-of-n oblivious transfer schemes. *Journal of Universal Computer Science*, 14(3).

Connolly, R. (2007). Entropy and random number generators in linux. *NIST Special Publication*.

Courtland, R. (2017). Google aims for quantum computing supremacy. *IEEE Spectrum*, 54(6):9–10.

Cret, O., Suciu, A., and Gyorfi, T. (2008). Practical issues in implementing trngs in fpgas based on the ring oscillator sampling method. In *10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2008. SYNASC'08.*, pages 433–438. IEEE.

Danger, J.-L., Guilley, S., and Hoogvorst, P. (2009). High speed true random number generator based on open loop structures in fpgas. *Microelectronics journal*, 40(11):1650–1656.

David, B., Dowsley, R., and Nascimento, A. C. (2014). Universally composable oblivious transfer based on a variant of lpn. In *International Conference on Cryptology and Network Security*, pages 143–158. Springer.

de la Fraga, L. G., Torres-Pérez, E., Tlelo-Cuautle, E., and Mancillas-López, C. (2017). Hardware implementation of pseudo-random number generators based on chaotic maps. *Nonlinear Dynamics*, 90(3).

Delvaux, J. and Verbauwhede, I. (2013). Side channel modeling attacks on 65nm arbiter pufs exploiting cmos device noise. *Hardware-Oriented Security and Trust (HOST)*.

Dyakonov, M. (2018). The case against quantum computing. https://spectrum.ieee.org/computing/hardware/the-case-against-quantum-computing.

EE Times (2010). Nxp and intrinsic-id to raise smart chip security. *UBM Tech Electronics*.

Elias, A. (2017). Understanding hardware roots of trust. https://www.synopsys.com/designware-ip/technical-bulletin/understanding-hardware-roots-of-trust-2017q4.html.

Feynman, R. P. (1959). There's plenty of room at the bottom [data storage]. *Journal of microelectromechanical systems*, 1(1):60–66.

Fiorin, L., Silvano, C., and Sami, M. (2007). Security aspects in networks-on-chips: Overview and proposals for secure implementations. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2007. DSD 2007*, pages 539–542.

Forte, D., Perez, R., Kim, Y., and Bhunia, S. (2016). Supply-chain security for cyberinfrastructure [guest editors' introduction]. *Computer*, 49(8):12–16.

François, M., Grosges, T., Barchiesi, D., and Erra, R. (2014). Pseudo-random number generator based on mixing of three chaotic maps. *Communications in Nonlinear Science and Numerical Simulation*, 19(4).

Gao, S. (2003). A new algorithm for decoding reed-solomon codes. In *Communications, Information and Network Security*, pages 55–68. Springer.

Gassend, B., Clarke, D., Van Dijk, M., and Devadas, S. (2002a). Controlled physical random functions. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, pages 149–160. IEEE.

Gassend, B., Dijk, M. V., Clarke, D., Torlak, E., Devadas, S., and Tuyls, P. (2008). Controlled physical random functions and applications. *ACM Transactions on Information and System Security (TISSEC)*.

Gassend, B. et al. (2002b). Silicon physical random functions. *Proceedings of the Computer and Communications Security Conference*.

Goldfeder, S., Chase, M., and Zaverucha, G. (2016). Efficient post-quantum zero-knowledge and signatures. *IACR Cryptology ePrint Archive*, 2016:1110.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM.

Günlü, O., Kernetzky, T., İşcan, O., Sidorenko, V., Kramer, G., and Schaefer, R. (2018). Secure and reliable key agreement with physical unclonable functions. *Entropy*, 20 (5):340.

Guo, C. and Chang, C.-C. (2013). Chaotic maps-based password-authenticated key agreement using smart cards. *Communications in Nonlinear Science and Numerical Simulation*, 18(6):1433–1440.

Herder, C., Yu, M.-D., Koushanfar, F., and Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141.

Hsu, J. (2018). Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy. *IEEE Spectrum Tech Talk*.

Hwang, J.-Y., bum Suh, S., Heo, S.-K., Park, C.-J., Ryu, J.-M., Park, S.-Y., and Kim, C.-R. (2008). Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In *5th IEEE Consumer Communications and Networking Conference, 2008. CCNC 2008.*, pages 257–261.

Iftene, S. (2007). General secret sharing based on the chinese remainder theorem with applications in e-voting. *Electronic Notes in Theoretical Computer Science*, 186:67–84.

Intel (2017). Intel security essentials. https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/security-essentials-solution-brief.pdf.

International Defense Security and Technology (IDST) (2019). Threats to ict supply chains including counterfeit electronic components and hardware trojans present critical risk to military systems, researchers developing new innovations. www.idstch.com/home5/international-defence-security-and-technology/cyber/the-digital-video-guard-can-protect-against-cyber-threats-of-hardware-trojans-claims-dsto-researchers.

Jae W, L., Devadas, S., et al. (2004). A technique to build a secret key in integrated circuits for identification and authentication applications. *VLSI Circuits*.

Jakimoski, G. and Kocarev, L. (2001). Chaos and cryptography: block encryption ciphers based on chaotic maps. *Ieee transactions on circuits and systems i: fundamental theory and applications*, 48(2):163–169.

Jerger, N. E. and Peh, L.-S. (2009). On-chip networks. *Synthesis Lectures on Computer Architecture*, 4(1):1–141.

Katz, J. and Malka, L. (2010). Secure text processing with applications to private dna matching. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 485–492. ACM.

Katz, J. and Shin, J. S. (2005). Modeling insider attacks on group key-exchange protocols. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 180–189, New York, NY, USA. ACM.

Kaufman, D. (2011). Darpa: Cyber analytical framework. https://www.slideshare.net/scovetta/darpa-cyber-analytical-framework-kaufman.

Khandelwal, S. (2015). Millions of iot devices using same hard-coded crypto keys. https://thehackernews.com/2015/11/iot-device-crypto-keys.html.

Khoshroo, S. (2013). Design and evaluation of fpga-based hybrid physically unclonable functions. *The University of Western Ontario*.

Kim, M., Ha, U., Lee, K. J., Lee, Y., and Yoo, H.-J. (2017). A 82-nw chaotic map true random number generator based on a sub-ranging sar adc. *IEEE Journal of Solid-State Circuits*, 52(7):1953–1965.

Kinsy, M., Bu, L., Isakov, M., and Mark, M. (2018). Designing secure heterogeneous multicore systems from untrusted components. *Cryptography*, 2(3):12.

Kinsy, M. A., Cho, M. H., Wen, T., Suh, G. E., van Dijk, M., and Devadas, S. (2009). Application-aware deadlock-free oblivious routing. In *36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA*, pages 208–219.

Kinsy, M. A., Khadka, S., Isakov, M., and Farrukh, A. (2017). Hermes: Secure heterogeneous multicore architecture design. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 14–20. IEEE.

Kinsy, M. A., Pellauer, M., and Devadas, S. (2011). Heracles: Fully synthesizable parameterized mips-based multicore system. In *International Conference on Field Programmable Logic and Applications, FPL 2011, September 5-7, Chania, Crete, Greece*, pages 356–362.

Kinsy, M. A., Pellauer, M., and Devadas, S. (2013). Heracles: A tool for fast rtl-based design space exploration of multicore processors. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 125–134, New York, NY, USA. ACM.

Knight, W. (2017). Ibm raises the bar with a 50-qubit quantum computer. *Sighted at MIT Review Technology*.

Kohlbrenner, P. and Gaj, K. (2004). An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78. ACM.

Kumar, R., Tullsen, D. M., Ranganathan, P., Jouppi, N. P., and Farkas, K. I. (2004). Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pages 64–75.

Kwok, H. and Tang, W. K. (2007). A fast image encryption system based on chaotic maps with finite precision representation. *Chaos, solitons & fractals*, 32(4):1518–1529.

Kwok, S. H. and Lam, E. Y. (2006). Fpga-based high-speed true random number generator for cryptographic applications. In *TENCON 2006. 2006 IEEE Region 10 Conference*, pages 1–4. IEEE.

Lapedus, M. (2018). A crisis in dod's trusted foundry program? www.semiengineering.com/a-crisis-in-dods-trusted-foundry-program.

Li, B., Li, H., Xu, G., and Xu, H. (2005). Efficient reduction of 1 out of n oblivious transfers in random oracle model. *IACR Cryptology ePrint Archive*, 2005:279.

Lim, D., Lee, J. W., Gassend, B., Suh, G. E., Van Dijk, M., and Devadas, S. (2005). Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205.

Liu, J., Juuti, M., Lu, Y., and Asokan, N. (2017). Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631. ACM.

Lynnyk, V., Sakamoto, N., and Čelikovskỳ, S. (2015). Pseudo random number generator based on the generalized lorenz chaotic system. *IFAC-PapersOnLine*, 48(18):257–261.

Lyubashevsky, V., Peikert, C., and Regev, O. (2010). On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer.

Mahmoud, A., Rührmair, U., Majzoobi, M., and Koushanfar, F. (2013). Combined modeling and side channel attacks on strong pufs. *IACR Cryptology ePrint Archive*, 2013:632.

Maiti, A., Kim, I., and Schaumont, P. (2012). A robust physical unclonable function with enhanced challenge-response set. *IEEE Transactions on Information Forensics and Security*, 7(1):333–345.

Majzoobi, M., Koushanfar, F., and Potkonjak, M. (2008). Lightweight secure pufs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673. IEEE Press.

Mandelbaum, R. F. (2018). This could be the best quantum computer yet. www.gizmodo.com/this-could-be-the-best-quantum-computer-yet-1831085617.

Martin, H., Martin-Holgado, P., Peris-Lopez, P., Morilla, Y., and Entrena, L. (2018). On the entropy of oscillator-based true random number generators under ionizing radiation. *Entropy*, 20(7):513.

Michael, M. H., Silveri, M., Brierley, R., Albert, V. V., Salmilehto, J., Jiang, L., and Girvin, S. M. (2016). New class of quantum error-correcting codes for a bosonic mode. *Physical Review X*, 6(3):031006.

Miller, J. E., Kasture, H., Kurian, G., Gruenwald, C., Beckmann, N., Celio, C., Eastep, J., and Agarwal, A. (2010). Graphite: A distributed parallel simulator for multicores. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12.

Morozov, S., Maiti, A., and Schaumont, P. (2010). An analysis of delay based puf implementations on fpga. In *International Symposium on Applied Reconfigurable Computing*, pages 382–387. Springer.

Nguyen, P. H., Sahoo, D. P., Jin, C., Mahmood, K., Rührmair, U., and van Dijk, M. (2018). The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *IACR Cryptology ePrint Archive*, 2018:350.

NIST (2012). True randomness can't be left to chance: Why entropy is important for information security. *NIST Special Publication*.

NIST (2018). Root of trust. csrc.nist.gov/Projects/Hardware-Roots-of-Trust.

NIST (2019). Post-quantum cryptography. csrc.nist.gov/news/2019/pqc-standardization-process-2nd-round-candidates.

Oberg, J., Sherwood, T., and Kastner, R. (2013). Eliminating timing information flows in a mix-trusted system-on-chip. *IEEE Design & Test*, 30(2):55–62.

Oder, T. and Güneysu, T. (2017). Implementing the newhope-simple key exchange on low-cost fpgas. *Progress in Cryptology–LATINCRYPT*, 2017.

Ohkubo, M., Miura, F., Abe, M., Fujioka, A., and Okamoto, T. (1999). An improvement on a practical secret voting scheme. In *International Workshop on Information Security*. Springer.

Özkaynak, F. (2014). Cryptographically secure random number generator with chaotic additional input. *Nonlinear Dynamics*, 78(3):2015–2020.

Pellauer, M., Adler, M., Kinsy, M. A., Parashar, A., and Emer, J. S. (2011). Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing. In *17th International Conference on High-Performance Computer Architecture (HPCA-17 2011), February 12-16 2011, San Antonio, Texas, USA*, pages 406–417.

Pogromsky, A. Y., Santoboni, G., and Nijmeijer, H. An ultimate bound on the trajectories of the lorenz system and its applications. *Nonlinearity*, 16.5:55–62.

Pöppelmann, T. and Güneysu, T. (2012). Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *International Conference on Cryptology and Information Security in Latin America*, pages 139–158. Springer.

Porquet, J., Greiner, A., and Schwarz, C. (2011). Noc-mpu: A secure architecture for flexible co-hosting on shared memory mpsocs. *Design, Automation and Test in Europe Conference and Exhibition*, pages 1–4.

Quantum Computing Report (2017). Applying moore's law to quantum qubits. www.quantumcomputingreport.com/our-take/applying-moores-law-to-quantum-qubits.

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34.

Rührmair, U., Sölter, J., Sehnke, F., Xu, X., Mahmoud, A., Stoyanova, V., Dror, G., Schmidhuber, J., and Burleson, W. (2013). Puf modeling attacks on simulated and silicon data. *IEEE Transactions on Information Forensics and Security*, 8.11:1876–1891.

Roy, S. S., Vercauteren, F., Mentens, N., Chen, D. D., and Verbauwhede, I. (2014). Compact ring-lwe cryptoprocessor. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 371–391. Springer.

Sajeesh, K. and Kapoor, H. K. (2011). An authenticated encryption based security framework for noc architectures. In *Proceedings of the 2011 International Symposium on Electronic System Design*, ISED '11, pages 134–139, Washington, DC, USA. IEEE Computer Society.

Salmani, H. (2018). The global integrated circuit supply chain flow and the hardware trojan attack. In *Trusted Digital Circuits*, pages 1–11. Springer.

Science-Daily (2018). World-first quantum computer simulation of chemical bonds using trapped ions. www.sciencedaily.com/releases/2018/07/180724110028.htm.

Seznec, A. and Sendrier, N. (2003). Havege: A user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 13(4):334–346.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22.11.

Shimizu, K., Suzuki, D., and Kasuya, T. (2012). Glitch puf extracting information from usually unwanted glitches. *IEICE Transactions on Fundamentals of Electronics*, 95.1:223 – 233.

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332.

(SIA), S. I. A. (2018). Beyond borders report. www.semiconductors.org/resources/beyond-borders-the-global-semiconductor-value-chain/sia-beyond-borders-report-final-may-6-1.

Soybali, M., Ors, B., and Saldamli, G. (2011). Implementation of a puf circuit on a fpga. *IEEE 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*.

Suh, E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. *Proceedings of the 44th annual Design Automation Conference*.

Sunar, B., Martin, W. J., and Stinson, D. R. (2007). A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on computers*, 56(1).

Tehranipoor, M. and Koushanfar, F. (2010). A survey of hardware trojan taxonomy and detection. *IEEE Design Test of Computers*, 27(1):10–25.

Tiwari, M., Wassel, H. M., Mazloom, B., Mysore, S., Chong, F. T., and Sherwood, T. (2009). Complete information flow tracking from the gates up. *ACM Sigplan Notices*, 37(1):109–120.

Tsoi, K. H., Leung, K., and Leong, P. H. W. (2003). Compact fpga-based true and pseudo random number generators. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 51–61.

Tzeng, W.-G. (2004). Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Transactions on Computers*, 53(2):232–240.

Vassilev, A. and Hall, T. A. (2014). The importance of entropy to information security. *Computer*, 47(2).

Wang, X., Tehranipoor, M., and Plusquellic, J. (2008). Detecting malicious inclusions in secure hardware: Challenges and solutions. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 15–19.

Wang, Y., Liu, Z., Ma, J., and He, H. (2016a). A pseudorandom number generator based on piecewise logistic map. *Nonlinear Dynamics*, 83(4):2373–2391.

Wang, Z., Karpovsky, M., and Bu, L. (2016b). Design of reliable and secure devices realizing shamir's secret sharing. *IEEE Transactions on Computers*, 65(8):2443–2455.

Wang, Z. and Karpovsky, M. G. (2011). Algebraic manipulation detection codes and their applications for design of secure cryptographic devices. *IEEE On-Line Testing Symposium*.

Wang, Z., Karpovsky, M. G., and Kulikowski, K. (2009). Replacing linear hamming codes by robust nonlinear codes results in a reliability improvement of memories. *IEEE/IFIP International Conference on Dependable Systems and Networks*.

Wassel, H. M. G., Gao, Y., Oberg, J. K., Huffmire, T., Kastner, R., Chong, F. T., and Sherwood, T. (2013). Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 583–594, New York, NY, USA. ACM.

Wieczorek, P. Z. and Golofit, K. (2014). Dual-metastability time-competitive true random number generator. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(1):134–145.

Wilson, L. (2016). The rise of quantum computers - the current state of cryptographic affairs. www.activecyber.net/rise-quantum-computers-current-state-cryptographic-affairs.

Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. (1995). The splash-2 programs: characterization and methodological considerations. In *Proceedings 22nd Annual International Symposium on Computer Architecture*, pages 24–36.

Xilinx (2016). Vivado design suite user guide - using constraints. *Xilinx Software Manuals*.

Xilinx (2018a). Vivado design suite user guide - implementation. *Xilinx Software Manuals*.

Xilinx (2018b). Vivado design suite user guide - synthesis. *Xilinx Software Manuals*.

Yu, M.-D., Hiller, M., Delvaux, J., Sowell, R., Devadas, S., and Verbauwhede, I. (2016). A lockdown technique to prevent machine learning on pufs for lightweight authentication. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):146–159.

Yu, M.-D. M., M'Raihi, D., Sowell, R., and Devadas, S. (2011). Lightweight and secure puf key storage using limits of machine learning. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 358–373. Springer.

Zorz, Z. (2018). Supply chain compromise: Adding undetectable hardware trojans to integrated circuits. https://helpnetsecurity.com/2018/12/10/hardware-trojans.

CURRICULUM VITAE