2020

Design, implementation, and analysis of efficient tools based on PUFs for hardware security applications

Hala Hamadeh
Iowa State University

Design, implementation, and analysis of efficient tools based on PUFs for hardware security applications

by

Hala Hamadeh

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering (Secure and Reliable Computing)

Program of Study Committee:
Akhilesh Tyagi, Major Professor
Ahmed Kamal
Doug Jacobson
Soma Chaudhuri
Swamy Ponpandi

Iowa State University

Ames, Iowa

2020

DEDICATION

I dedicate this work to my loving husband Abdallah who has been a constant source of support and unconditional love, and without his support I would not have been able to complete this work. To my kids Diyaa and Yusuf, who have been a cheerleader in my graduate studies. To my family and friends for their constant support. Lastly, to my mom, Salam and my father Mohammad, who always encouraged me and prayed for my success.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ALU	Arithmetic Logic Unit
AIK	Attestation Identity Key
CFG	Control Flow Graph
CRPs	Challenge Response Pairs
CPU	Central Processing Unit
DFM	Design for Manufacturability
DSA	Directed Self Assembly
ECC	Elliptic-Curve Cryptography
FHE	Fully-Homomorphic Encryption
FPGA	Field-Programmable Gate Array
IC	Integrated Circuits
IoTs	Internet of Things
IP	Intellectual Property
LEs	Logic Elements
MTR	Metadata Tracking Register
NIST	Institute of Standard and Technology

PCPs	Probabilistically Checkable Proofs
PCR	Platform Configuration Register
PUF	Physical Unclonable Function
RCA	Ripple Carry Adder
RNG	Random Number Generator
RNS	Residue Number System
RSSI	Received Signal Strength Indicator
RTS	Root of Trust for Storage
TCG	Trusted Computing Group
TGs	Transmutation Gates
TPM	Trusted Platform Module
VC	Verifiable computations
VLSI	Very-Large-Scale Integration

ACKNOWLEDGMENTS

In the name of Allah, I would like to thank Allah for giving me the strength, knowledge, ability and opportunity to complete this thesis.

I would like to take this opportunity to express my sincere gratitude to those who helped me with various aspects of research and the writing of this thesis. First and foremost, I'm extremely grateful to Dr. Akhilesh Tyagi for his guidance, patience that cannot be underestimated, and unwavering support throughout the past seven years. He has constantly steered my work towards being substantial. Dr. Tyagi has supported me not only academically but also emotionally through the rough road to finish this thesis.

I would also like to thank my committee members for the input and sense of direction they provided: Dr. Ahmed Kamal, Dr. Soma Chaudhuri, Dr. Swamy Ponpandi, Dr. Mohamed Selim and Dr. Doug Jacobson.

I am deeply indebted to my best friend, soul-mate, my husband Abdallah who has always been a true and great supporter and has unconditionally loved me during my good and bad times.

I would like to express my deepest gratitude to my brothers and sisters (Alaa, Fedaa, Reem, Yazan, Haneen, Laith, and Ibrahim) for providing their support and always encouraging me.

Of course, last but not least, I thank my parents who have supported and encouraged me in every way, and who raised me with the skills to succeed and never give up.

ABSTRACT

A Physical Unclonable Function (PUF) is a physical system that leverages manufacturing process variations to generate unclonable and inherent instance-specific measurements of physical objects. PUF is equivalent to human biometrics in many ways where each human has a unique fingerprint. PUF can securely generate unique and unclonable signatures that allow PUF to bootstrap the implementation of various physical security issues. In this thesis, we discuss PUFs, extend it to a novel SW-PUF, and explore some techniques to utilize it in security applications.

We first present the “SW-PUF” - basic building block of this thesis, a novel PUF design that measures processor chip ALU silicon biometrics in a manner similar to all PUFs. Additionally, it composes the silicon measurement with the data-dependent delay of a particular program instruction in a way that is difficult to decompose through a mathematical model. We then implement the proposed PUF to solve various security issues for applications such as Software Protection and Trusted Computing. We prove that the SW-PUF can provide a more robust root of trust for measurement than the existing trusted platform module (TPM).

Second, we present the “Reversible SW-PUF”, a novel PUF design based on the SW-PUF that is capable of computing partial inputs given its outputs. Given the random output signature of specific instruction in a specific basic block of the program, only the computing platform that originally computed the instruction can accurately regenerate the inputs of the instruction correctly within a certain number of bits. We then implement the Reversible SW-PUF to provide a verifiable computation method. Our scheme links the outsourced software with the cloud-node hardware to provide proof of the computational integrity and the resultant correctness of the results with high probability.

Finally, we employ the SW-PUF and the Reversible SW-PUF to provide a trust attribute for data on the Internet of Thing (IoT) systems by combining data provenance and privacy-preserving methods. In our scheme, an IoT server can ensure that the received data comes from the IoT device that owns it. In addition, the server can verify the integrity of the data by validating the provenance metadata for data creation and modification.

CHAPTER 1. INTRODUCTION AND BACKGROUND

In an ideal world where hackers don't exist, computer security is not an issue. While computer and mobile device users can simply and blindly use their devices to perform a variety of sensitive tasks (e.g., accessing bank accounts, paying bills, or outsourcing computation), in the real world maintaining trust and security is, unfortunately, a very challenging task, with a new software vulnerability reported every day. Scholars continuously propose new security solutions while hackers continuously try to find a back door for overcoming these solutions. This battle between scholars and hackers every day keeps the door open for a better security system. With the rapid development of the new technology, maintaining security and trust for users' sensitive data is even more important.

A Physical Unclonable Function (PUF) is employed to solve various hardware security issues, such as authentication, secret key generation, and software licensing. However, the traditional PUFs are not well-suited for many emerging security issues because they are vulnerable to observe once, run everywhere (OORE) attacks. The work in this thesis explores robust PUFs that compose the silicon measurement with the data-dependent delay of a particular program instruction in a way that is difficult to decompose through a mathematical model. This feature of the PUF is employed to provide various security solutions.

In the first part of this thesis, we propose a novel SW-PUF design approach. Then we employ the proposed PUF to provide reliable and secure platforms that offer more robust bases of measurement trust than the existing trusted platform module (TPM). Next, we extend our PUF design to a reversible SW-PUF, where the new reverse computing capability provides an efficient mechanism for verifiable computations in the cloud computing domain.

In the last part of this thesis, we present a privacy-preserving data provenance solution that merges the proposed PUF with non-interactive zero-knowledge proof to provide trustworthy and dependable IoT systems.

1.1 Background

This Section reviews background and prior work related to the topic of this thesis that are going to be presented in the following chapters. A brief background of PUF types and their applications is introduced in Section 1.1.1. Section 1.1.2 reviews the definition and some prior work on Trusted Computing issues. A definition of the Verifiable Computations and some prior work is presented in section 1.1.3. Finally, section 1.1.4, introduces an overview of the Data Provenance.

1.1.1 Physical Unclonable Function overview

In Integrated Circuits (IC) technology many parameters of chip production are not fully controlled. Timing parameters, for example, are sensitive to process variations that can be caused by factors such as contamination, metal and oxide thickness variations. These factors result in small variations for each IC; meaning that each chip is unique and no two physical objects are exactly the same even if they have been fabricated under the same conditions. However, such variations are usually not significant enough to affect chip performance. To facilitate the extraction of such unique physical characteristics, a physically unclonable function could be implemented. A PUF is a mathematical function that is derived from a physical system to generate unique signatures (responses R_i) to a corresponding (challenges C_i), where the challenge-response relation is defined by a process variability of semiconductor devices (Kawa et al. (2006)). Typical PUF responses are random, unpredictable and almost impossible to reproduce. The applications of PUFs mainly relate to physical security tasks of a low-cost, resource-constrained electronic device.

1.1.1.1 Classification of PUFs

Many types of PUFs have been presented in the literature, PUFs can be categorized based on different properties. For example, PUFs can be classified based on operation and construction principles into three classes: silicon PUFs, electronic PUFs, and non-electronic PUFs.

Silicon PUFs

Silicon PUFs are implanted on a silicon IC chip, they exploit the unmanageable CMOS manufacturing process variations to generate a unique stamp to characterize each IC chip. There are two major types of silicon PUFs: Delay-based silicon PUFs and memory-based silicon PUFs. The most popular delay-based PUFs are the Ring Oscillator PUFs which were proposed for the first time by Gassend (2003) and the arbiter PUFs which were introduced by Lim et al. (2005).

Figure 1.1 shows the basic implementation of the Ring Oscillators (RO) circuit. The basic structure consists of several identical ring oscillators, an n-to-2 multiplexer, two counters, and a comparator. Each of the identical ROs oscillates has a different frequency due to manufacturing process variations. To generate an unpredictable sequence of bits, one pair of ring oscillators is selected then the counters will count the oscillation cycles for a certain period of time.

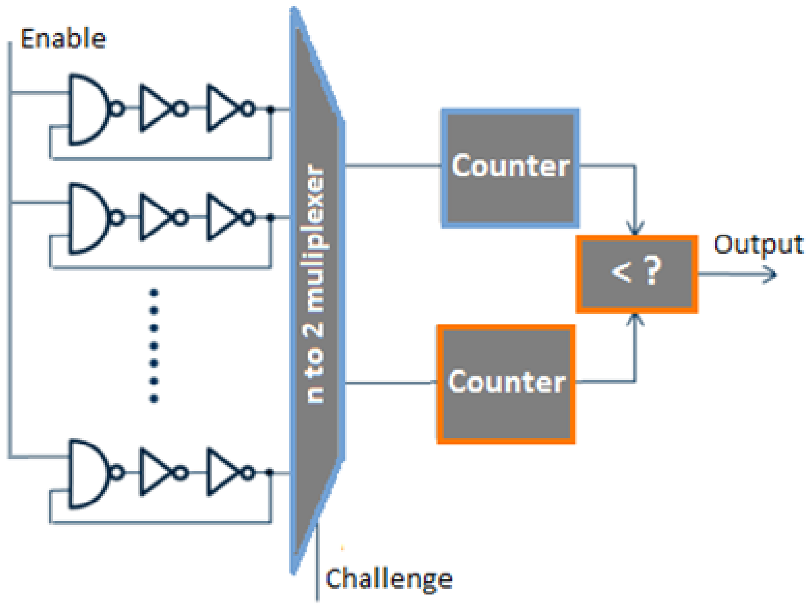


Figure 1.1 Ring Oscillator PUF basic structure.

Electronic PUFs

Electronic PUFs are based on analog measurements of electric properties of an object such as power, resistance, and capacitance. Coating PUF - which has been proposed by Skoric et al. (2007) - is one example of electronics PUFs. It can be implemented on the top metal layer of an IC chip and the randomness of capacitance measurements between each pair of metal wires will be employed to generate a unique identifier for each IC chip.

Non-electronic PUFs

Non-electronic PUFs involve all functions based on non-electric technologies or materials. However, most of the time, an electronic and digital techniques will be employed to process the challenge /response pairs. The non- electronic PUFs are the oldest types of PUFs and are usually based on random reflection of optical fibers or lasers. A popular example of non-electronic PUF is the Optical PUF as proposed by Pappu et al. (2002a) . Figure 1.2 illustrates the basic implementation of an Optical PUF, which consists of a transparent material such as glass. When a coherent laser beam shines on the material, a totally random and unique pattern could be and recorded. This pattern is almost impossible to reproduced because it depends on the position of the scattering particles and the characteristics of the laser beam such as wavelength, angle, and location.

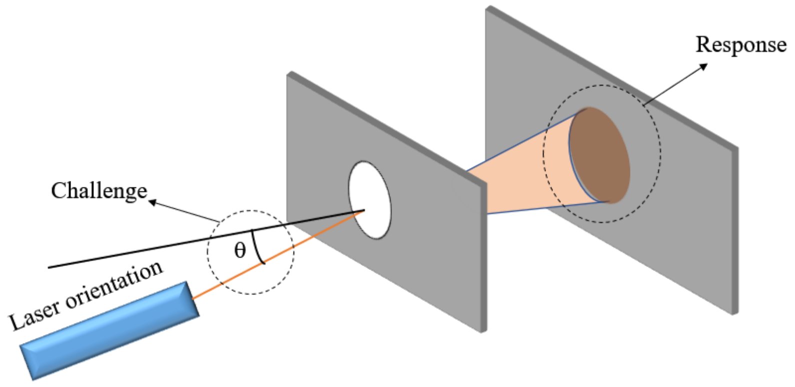


Figure 1.2 Optical PUF basic structure.

1.1.1.2 Properties of PUFs

There are a wide variety of properties for PUF that have been proposed in the literature - Roel (2012). Not all of these properties are required to define an acceptable PUF. Each set of specific properties can be useful in a specific application and the remaining properties can be nice-to-have qualities. In this section we will only list the most important properties that are required for physical security solutions, which are:

- Unique: The responses of the PUF must include some unique information about the identity of the actual physical entity.
- Unclonable: It is the core property of a PUF. This means that it should not be possible to clone the exact physical entity even if it was manufactured in the same production process as the original.
- Reproducible: The response of the PUF must be reproducible up to a small error.
- Unpredictable: The response of the PUF must be hard to predict giving only the challenge.

1.1.1.3 Evaluations of PUFs

There are three Common metrics to evaluate the PUFs responses:

- Uniqueness: Uniqueness measures the capability to distinguish between different devices. Hamming Distances (HD) between PUF responses are used to measure uniqueness. An ideal HD between any two PUF responses is 50%.
- Randomness: Randomness evaluates two properties in a PUF signature by analyzing the distribution of 0's and 1's, the independence and the uniformity.
- Reliability: Reliability measures robustness of a PUF in the presence of environmental variations. Temperature variations are the main factor that affect the stability of a PUF response.

1.1.1.4 Applications of PUFs

PUFs can be deployed in many applications based on their properties. Providing a physical security solution is one of the most popular applications. This could be achieved by deploying an appropriate PUF in a scheme together with other software or hardware to improve its characteristics. Very broad application based on PUFs have been proposed in the past staring with applications that require some sort of randomness such as random number generators, and cryptographic key generation Aldaghri and Mahdavifar (2018). PUFs also seem to be an elegant solution for application that require device authentication where some device property can be obtained from process variation as proposed by Suh and Devadas (2007a). The authors in Guo et al. (2018) have also employed PUFs in Cryptographic applications.

1.1.2 Trusted Computing

Trusted computing refers to hardware components and software technologies that together provide confidence that a computing platform will operate as expected. In other words, trusted computing can solve computer security problems using additional hardware. A group of international industry hardware manufacturers and software vendors had formed a not-for-profit organization called the Trusted Computing Group (TCG) for developing some policies and specifications to combining trusted hardware with various computing platforms. These specifications help protect data, hardware, and other resources from stealing, damage, or compromise without adversely impacting the rights of participating individuals or businesses. A Trusted Platform Module (TPM) is a major building block in achieving the goals of a TCG. The TPM uses hardware and associated software to provide a trusted computing base.

1.1.2.1 Trusted Platform Module

Trusted Platform Module (TPM) is a hardware chip that is embedded within a trusted computing platform to provide a hardware root of trust. An asymmetric public, private key pair embedded into TPM serves to prove its unique identity. The TPM can securely store cryptographic keys which can be used to authenticate the platform and protect its information. The TPM's design relies on the host platform for software measurements. It receives the measurements of the platform components performed by measuring software started on the CPU at boot stage. It stores these values in Platform Configuration Registers (PCRs) then extends these values at each stage. Thus, the TPM serves as a root of trust for reporting the measurement rather than performing the actual measurement. The TPM is able to hash and report the actual measurements from an untrusted environment. It is impossible for the TPM to verify these measurements. Figure 1.3 shows the major components of a TPM.

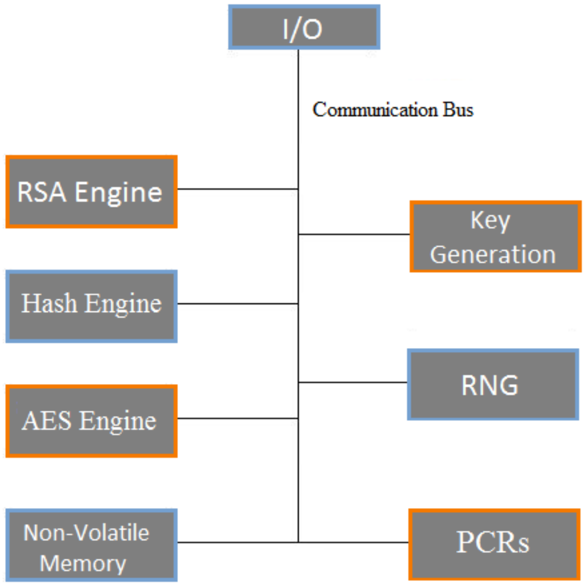


Figure 1.3 TPM major components.

Platform Configuration Register (PCR)

Platform Configuration Registers (PCRs) are one of the essential features of a TPM; they are used to store hash digests that represent platform integrity measurements. PCRs are shielded inside the TPM where data is protected against interference and exposure. PCRs are designed to allow a secure representation of the host system's configuration metric, which can be used to monitor any changes in the software and hardware configurations. The hash value of any PCR is updated by concatenating a new digest value with the original digest value, followed by a hash operation whose result is stored back to the PCR.

1.1.3 Verifiable Computations

Cloud computing has had significant growth in recent years, which attracted interest in the domain of Verifiable Computation (VC). Cloud computing is a popular choice for many individuals and small businesses for all the benefits such as enhanced productivity, increased efficiency, and saving money. Computation and resource starved mobile devices also tend to perform their heavy-duty computation on the cloud. However, the security and the correctness of cloud computing are a critical concern due to the sensitivity and importance of the data outsourced to the Cloud. The concept of verifiable computation allows a lower-resource client to safely outsource the computation of a program to an untrusted cloud. This cloud performs the computation and provides a proof asserting their correctness, the client can verify that the produced results are consistent with the program specification and that the computations were performed correctly. To be viable, the effort of performing the verification must be negligible compared to the actual computation.

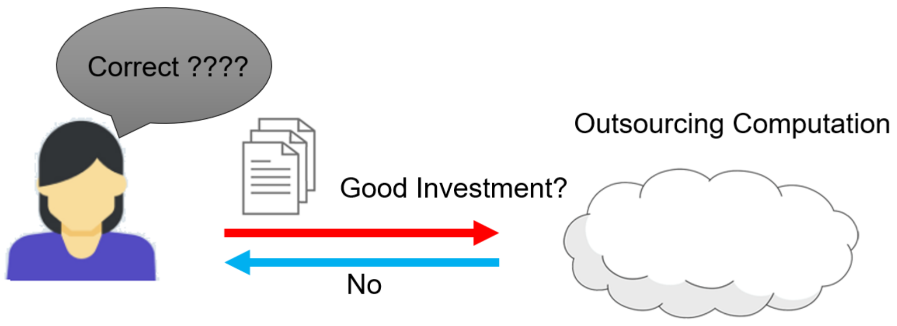


Figure 1.4 Verifiable computation scenario.

Three main solutions were proposed in the literature to support verifiable computation: Verifiable Computation based on Trusted Computing as proposed by Sailer et al. (2004), and Chen et al. (2006). This approach depends on some trusted hardware such as TPM. The main drawback of this solution was the assumption that the physical protections cannot be defeated. The second solution is Verifiable Computation with a Non-Interactive Argument. It is described by Parno et al. (2013), and Parno et al. (2016). This approach relies on complex Probabilistically Checkable Proofs (PCPs) or Fully-Homomorphic Encryption (FHE). However, this theoretical solution is not practical because it requires thousands of years to be executed, even for a simple computation. The last solution is Verifiable Computation with Interactive Proofs which was proposed by Vu et al. (2013), and Thaler et al. (2012). This protocol involves an exchange of massive number of messages between the cloud and the client. During this conversation, the cloud aims to convince the client that the computation is performed correctly. While this approach is often efficient, it applies only to a narrow class of computations.

1.1.4 Data Provenance

Data provenance refers to the process of maintaining data integrity and authenticity by recording the data origin, entities, processes, and systems that impact data of interest. Data provenance describes the data life cycle and tracks the data as it goes through diverse processes. In this thesis, we are interested particularly in the data provenance protocols for IoT systems, where such systems have many resource-constraints.

Recently, several data provenance protocols have been investigated in literature. For instance, the protocol proposed by Sanchez et al. (2018) extended the IBM Idemix protocol with a non-interactive zero-knowledge proof to sign the metadata transmitted by an IoT device in a privacy-preserving way. However, this solution is limited only for some types of systems. Another work proposed by Alharbi and Lin (2012) investigated the privacy-preserving data provenance. However, their design relies on the trust of the server itself on an ecosystem. Few other techniques established a secure data provenance based on PUFs. The protocol proposed by Javaid et al. (2018) used PUF with blockchain network in order to add a unique hardware fingerprint of each IoT device. Their proposed solution depends on storing many Challenge Response Pairs (CRPs) in the server memory exposing them to storage attack concerns. Aman et al. (2017) have used the Received Signal Strength Indicator (RSSI) along with PUF and symmetric encryption to provide the privacy of the transmitted metadata between the IoT device and the server. Similar to the work proposed by Javaid et al. (2018), this scheme also relies on storing many challenge-response pairs (CRPs) in the server; in addition it also relies on the RSSI values which is not practical in some IoT environments.

1.2 Thesis Contributions

In this thesis, we make the following contributions:

- Propose a novel PUF design, "SW-PUF", that measures processor chip ALU silicon biometrics and composes them with the data-dependent delay of a particular program.
- Propose a protocol for third-party verification using the SW-PUF for measuring software activity at run-time to ensure the control-flow integrity of a program.
- Present a static root of trust scheme based on the proposed PUF that can offer more advanced protection than the TPM.
- A VLSI area and energy analyses are performed for a subset of TPM commands to compare the SW-PUF with TPM.
- Extend the SW-PUF design to make it reversible. The new PUF is able to compute partial inputs after being given its outputs.
- Develop a protocol based on the reversible SW-PUF based on the Bayesian method for Verifiable Computations in cloud computing within the class of interactive proof systems.
- evaluate our verifiable computation scheme to demonstrate that it yields faster verification than previous approaches.
- Present a privacy-preserving data provenance solution that merges the SW-PUF with non-interactive zero-knowledge proof to provide trustworthy and dependable IoT systems.

- Implement the proposed data provenance protocol on FPGA Altera Cyclone, for evaluating the scheme that was developed in this work.

1.3 Thesis Outline

This section outlines the structure of the thesis and provides a brief review of each chapter. The thesis is organized as follows:

Chapter 1: The first chapter briefly introduces the motivation of this research, and separately outlines the contributions of each chapter.

Chapter 2: In this chapter, we introduce the design, implementation and evaluation of a new type of PUFs. The proposed PUF is given the name of "SW-PUF". Then we developed and evaluated protocols based on this PUF to provide trusted computing and software integrity measurement solutions.

Chapter 3: In this chapter, we introduce the design, implementation and evaluation of a new type of PUFs, which we called "reversible SW-PUF". Then we proposed and evaluated a protocol based on this PUF to provide a Verifiable Computations base for cloud computing within the class of interactive proof systems.

Chapter 4: In this chapter, we developed and evaluated a privacy-preserving data provenance scheme that is based on the SW-PUF and the reversible SW-PUF to provide trustworthy and dependable IoT systems.

Chapter 5: In this chapter, we summarize the most important findings of this thesis and propose a number of future research directions.

1.4 References

Aldaghri, N. and Mahdavifar, H. (2018). Fast secret key generation in static environments using induced randomness. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE.

Alharbi, K. and Lin, X. (2012). Pdp: A privacy-preserving data provenance scheme. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 500–505. IEEE.

Aman, M. N., Chua, K. C., and Sikdar, B. (2017). Secure data provenance for the internet of things. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, pages 11–14. ACM.

Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R., and Stüble, C. (2006). A protocol for property-based attestation. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, STC '06, pages 7–16, New York, NY, USA. ACM.

Gassend, B. L. P. (2003). *Physical random functions.* PhD thesis, Massachusetts Institute of Technology.

Guo, Y., Dee, T., and Tyagi, A. (2018). Barrel shifter physical unclonable function based encryption. *Cryptography*, 2(3):22.

Javaid, U., Aman, M. N., and Sikdar, B. (2018). Blockpro: Blockchain based data provenance and integrity for secure iot environments. In *Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems*, pages 13–18. ACM.

Kawa, J., Chiang, C. C., and Camposano, R. (2006). EDA challenges in nano-scale technology. In *CICC*, pages 845–851. IEEE.

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.

Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, Washington, DC, USA. IEEE Computer Society.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2016). Pinocchio: Nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112.

Roel, M. (2012). Physically unclonable functions: Constructions, properties and applications. *Katholieke Universiteit Leuven, Belgium.*

Sailer, R., Zhang, X., Jaeger, T., and van Doorn, L. (2004). Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 16–16, Berkeley, CA, USA. USENIX Association.

Sanchez, J. L. C., Bernabe, J. B., and Skarmeta, A. F. (2018). Towards privacy preserving data provenance for the internet of things. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 41–46. IEEE.

Skoric, B., Schrijen, G.-J., Ophey, W., Wolters, R., Verhaegh, N., and van Geloven, J. (2007). Experimental hardware for coating pufs and optical pufs. In *Security with Noisy Data*, pages 255–268. Springer.

Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE.

Thaler, J., Roberts, M., Mitzenmacher, M., and Pfister, H. (2012). Verifiable computation with massively parallel interactive proofs. In *HotCloud*. USENIX Association.

Vu, V., Setty, S. T. V., Blumberg, A. J., and Walfish, M. (2013). A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 223–237. IEEE Computer Society.

CHAPTER 2. PHYSICAL UNCLONABLE FUNCTIONS (PUFS) ENTANGLED TRUSTED COMPUTING BASE

Modified from a paper accepted by *IEEE-iSES 2019 conference*

Hala Hamadeh and Akhilesh Tyagi

Best paper award winner

2.1 Abstract

The center-piece of this chapter is a software measurement physical unclonable function (PUF). It measures processor chip ALU silicon biometrics in a manner similar to all PUFs. Additionally, it composes the silicon measurement with the data-dependent delay of a particular program instruction in a way that is difficult to decompose through a mathematical model. This approach ensures that each software instruction is measured if computed. This constitutes a more robust root of trust for measurement than the existing trusted platform module (TPM). The SW-PUF measurements bind the execution of software to a specific processor with a corresponding certificate. This makes the SW-PUF a promising candidate for applications requiring Software Protection and Trusted Computing. For instance, it could measure the integrity of an execution path by generating a signature that is unique to the specific program execution path and the processor chip. To explore the feasibility of the proposed schemes, the SW-PUF have been implemented in HSPICEk-2015.06 using 45 nm technology. It is analyzed in term of three metrics: uniqueness, reliability, and randomness. Our proof-of-concept implementation shows good uniqueness compared to other types of PUFs, the average Hamming distance between a pair of responses over different software PUFs (different instruction input data) was 32% (50%). The SW-PUF exhibits more than 96% reliability for temperatures from -10°C to 65°C. The randomness was measured with NIST suite of 14 tests - wherein 12-13 tests were consistently passed with a minimum pass rate of 90.6%. A

Verilog synthesis based comparison of SW-PUF versus TPM area and energy shows that area needs of SW-PUF are less than 0.01% of TPM area needs and energy of SW-PUF is less than 0.0001% of TPM energy.

2.2 Introduction

The VLSI fabrication technology at small scale is statistical by nature. Many parameters of chip production are not fully controlled. Timing parameters, for example, are sensitive to process variations that can be caused by factors such as contamination, metal and oxide thickness variations, and lithography variations. These factors result in small variations in threshold voltage and gate oxide thickness for each logical gate; meaning that each chip is unique and no two transistors are identical in delay even if they have been fabricated under the same conditions. Such variations are usually not significant enough to affect circuit performance by stretching the clock period to hide these variations. The differences in process parameters can be exposed at sub-clock period level to generate a unique chip biometric identity to provide authentication for each chip. A Physical Unclonable Function (PUF) is a physical system to leverage such process variations to generate a response to a challenge Kawa et al. (2006). Traditional Challenge-Response Pair (CRP) PUFs are not well-suited for the software protection problem in offline settings as stated by Nithyanand and Solis (2012a) because they are vulnerable to observe once, run everywhere (OORE) attacks. Nithyanand and Solis (2012a) argue that the PUF measurements should be interleaved at a finer granularity than the whole program to achieve robust software protection.

Within a processor, the datapath elements (which are all silicon-based) are the function units that interact to perform the computation of a software instruction. Program inputs determine the execution path which determines what input data is received by each function unit. The delays of a function unit that can be attributed to the instruction input data values become in a loose manner the execution path identity. Combining these delays with silicon variation based delays in a non-decomposable manner will allow us to realize a PUF for composite identity of software and processor. We accomplish this goal of entangling the program instruction level inputs with function unit level silicon variability in a novel SW-PUF design that composes the data-dependent delay variation in a function unit with the silicon process level delay variation to measure both the software and the device. The measurement reflects both the program instruction input values and process level variation within the processor chip function unit. The function unit delay variation results from both the process variation (hardware dependent) and the function unit data inputs (software dependent). In a five-stage pipeline MIPS implementation, a typical software instruction-mix consists of the ALU, load/store, and control class of instructions. The ALU instructions use the ALU for the desired computation. The address computation of a load/store instruction uses an ALU unit, the control instructions also use the ALU unit for branch target address calculation. A datapath ALU design that automatically measures each instruction enables a root of trust for measurement to ensure that each instruction in the software is measured through the ALU. In our approach, the existing ALU unit is repurposed slightly to function as a virtual PUF as a side-effect.

The design of the SW-PUF has been evaluated in HSPICE using predictive technology model and tested on 8 FPGAs. The evaluation of this ALU PUF reveals excellent randomness, acceptable uniqueness, and satisfactory reliability. As in any traditional PUF, use of error correction is needed to ensure the reproducibility in its response. Herder et al. (2017) presented a good error correction scheme. However, in this work, the focus is on the design and implementation of the SW-PUF to assess its feasibility for software verification. The robustness properties such as reproducibility through error correction would be the target of future research.

The rest of the paper is organized as follows. In Section 2.3, we summarize the existing work on PUFs and describe relevant concepts. In section 2.4, we describe the design of the classic RCA. Section 2.5 presents the design of the proposed SW-PUF. In Section 2.6, we describe software integrity measurement framework. We then compare the SW-PUF with a TPM in Section 2.7. The experimental and simulation results are given for the SW-PUF in Section 2.8. Finally, Section 2.9 concludes the paper.

2.3 Related Work

Physical Unclonable Functions (PUFs) or Physical One-Way Functions (P-OWFs) were first proposed in 2002 by Pappu et al. (2002b). PUFs are mostly implemented based on timing and delay variation of silicon circuits. Our proposed design is also based on such timing parameters variability; however, other kinds of PUFs have also been proposed. For example, Guajardo et al. (2007a) proposed a Memory-Based PUF that uses SRAM memory randomness, while Pappu et al. (2002b) proposed a non-electronic optical PUF that depends on the speckle patterns of an optical medium under laser illumination. PUFs are suitable for use in many applications such as hardware authentication as presented by Suh and Devadas (2007b), Intellectual Property (IP) protection such as the work presented by Guajardo et al. (2007b), software protection presented by Nithyanand and Solis (2012b), cryptographic key generation presented by Guajardo et al. (2007b), and PUF-based RFID for anti-counterfeiting presented by Devadas et al. (2008). PUFs also have been used on a mobile device to support authentication presented by Scheel and Tyagi (2015).

There are several research efforts to build a PUF out of an ALU. Kong et al. proposed ALU PUF which is based on the delay difference in two identical ALUs along the lines of an arbiter PUF Kong et al. (2014). They proposed to combine the output of the ALU PUF with the checksum computation in a remote attestation mechanism to bind the computation to hardware ALU. However, their ALU PUF does not seem to have any program data-dependent properties tying it to the execution of a program - unlike this research. They used two modes for the ALU, one is the normal mode and the other is to perform the PUF operation.

PUFs are also used with TPM to enhance security. Choi and Kim (2012) proposed to protect the keys inside the TPM through the use of key-hiding with a PUF. In a similar work presented by Li et al. (2016), Chatterjee et al. (2018), the authors present approaches for enhancing the security of key hierarchy in a TPM using PUF. Lebedev et al. (2018) and Müller et al. (2018) proposed to use PUF in order to build a secure boot. However, all the existing TPM solutions suffer from a major problem since it is impossible for the TPM to verify the received measurements. Our approach differs from the TPM principally in that all the measurements occur within trusted boundaries inside the CPU.

2.4 Background

2.4.1 Ripple Carry Adder (RCA)

A ripple carry adder is constructed by chaining stages of full adders, with the carry output from each full adder rippling as a carry input to the next full adder. For simplicity, a 4-bit RCA is shown in Figure 2.1. The final sum of the most significant bit becomes valid after the carry signal has rippled all the way through the adder, from the least significant stage to the most significant stage, in the worst case. The worst-case delay through an n-bit RCA occurs when a carry signal ripples from the least significant bit to the most significant bit through all stages; its approximate value is given by the following Equation:

$$O(n) = (\sum_{i=0}^{n-1} tc) + ts \tag{2.1}$$

where tc is the delay through the carry stage of a single full adder and ts is the delay in computing the final correct sum. However, the average carry chain length is $O(\log n)$ when random input data is presented by Parhami (2009). Garside (1993) determined that the average carry propagation length was 4 for a 32-bit adder. Each full adder contains 5 gates: 1 OR, 2 ANDs, and 2 XORs, and every gate has a different propagation delay, so the total delay at each stage will be unique due to process level variations. The total delay of the adder depends on the carry chain length and the delay of each stage. Since these factors can be identical only if the inputs to the adder and the physical adder itself remain unchanged, we can state that this delay is unique for each set of input data and processor.

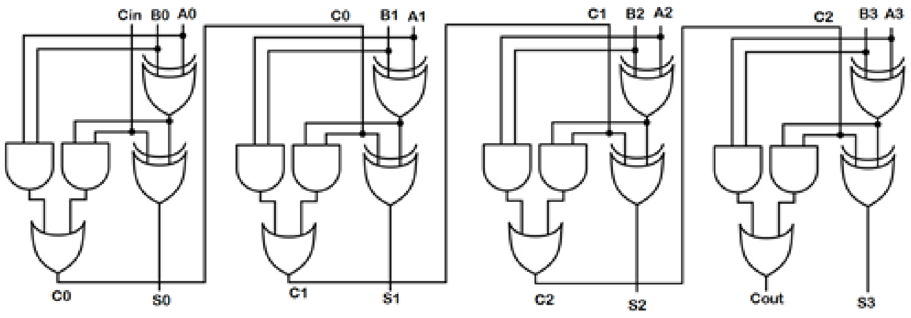


Figure 2.1 4-bits RCA circuit

2.5 The Principle of SW-PUF Design

The design of the SW-PUF aims to capture the delay variations caused by the fabrication process and input data. The proposed ALU schema is shown in Figure 2.2. In this design, the ALU output is sampled twice - once at the normal worst case clock period which is chosen to be long enough to hide transistor level delay variability and the data-dependent delay variability; while the shifted-clock is chosen to sample the ALU output early. The time period of the early-sampling shifted-clock should be less than the average of the total delay of the ALU, which is $\log n$ as stated by Burks et al. (1987) and Parhami (2009). This ensures that the instruction level data influences the SW-PUF output. Inverters are used to generate a slight time shift between the clock and the shifted-clock. The early sampled output is the SW-PUF output. The normal clock sampled output is the data output generated by the instruction for the normal program consumption. Note that the overhead of the proposed schema is minimal, one extra latch per output bit, in addition to the shifted clock generation overhead.

As shown in Figure 2.2, the proposed design is very simple to implement using an existing ALU, a D Flip-flop to latch the input data at time $\tau 1$, and a D Flip-flop to latch the output data at $\tau 2$, where:

$$\tau 2 = \tau 1 + t_{delay} \tag{2.2}$$

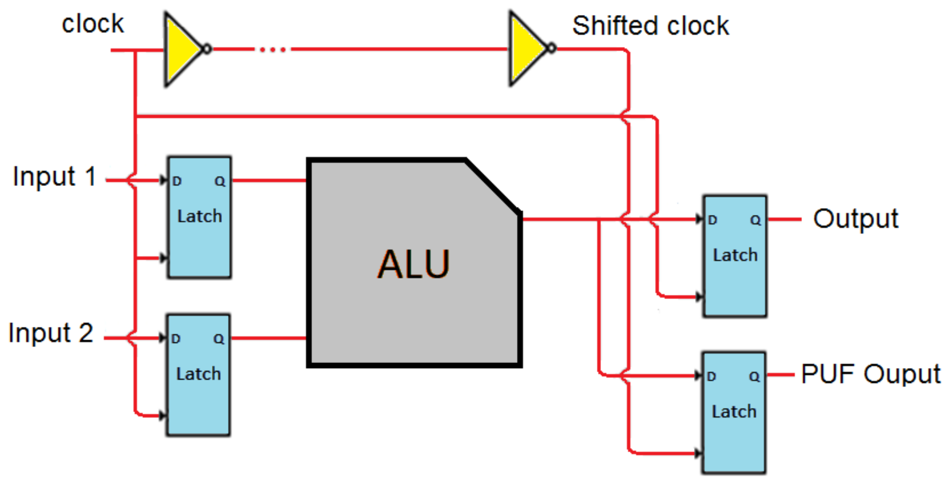


Figure 2.2 The proposed architecture of the SW-PUF

The clock edge at $\tau 2$ leads the clock edge at the normal delay ($\tau 1 + T$ for clock period T) for early output sampling. The output data will be likely random and "noisy" at this time. In this case, the output data of the ALU is a function of three parameters: the ALU inputs, the previous state of the circuit, and the transistor-level variability.

2.6 Software integrity measurement

Measurement is a process of characterizing software. The reasons for measuring a software are varied leading to a variety of measurement techniques. SW-PUF's strength is the guarantee that every instruction is measured in hardware. An instruction computation and its measurement are atomic. Such software instruction measurement sequences at run-time can generate a unique signature of a specific program execution path within its Control Flow Graph (CFG) on a *specific* processor chip.

An execution path resulting in the dynamic instruction sequence: $I0,\ I1,\ I2,\ \ldots,\ In$ generates a corresponding sequence of PUF stamps in the modified processor, $P0,\ P1,\ P2,\ \ldots,\ Pn$. Note that the same execution path could result in different PUF stamp sequences since the same instruction I_l (such as an `add r1, r2, r3`) could have different input data in `r2`, `r3` - especially when this data does not determine the control path for the execution. There are many ways this PUF stamp sequence can be used for software protection and certification. For instance, control flow integrity can be certified by creating a hash-based digest of the PUF stamp sequence similar to TPM's PCR extension as $D_0 = h(P0)$; $D_i = h(D_{i-1} \mid\mid Pi)$, where h is a hash function, $\mid\mid$ denotes concatenation, and D_i is the program signature digest until the instruction I_i. D_n is the unique certificate of the control flow and data flow that took place for the execution $I0, I1, I2, \ldots, In$. A third party can verify that the control flow indeed was correct by comparing such control path PUF stamp digest. Although, due to the measurements being inextricably bound to the platform, a verifying party would also need to obtain measurements from this original computing platform.

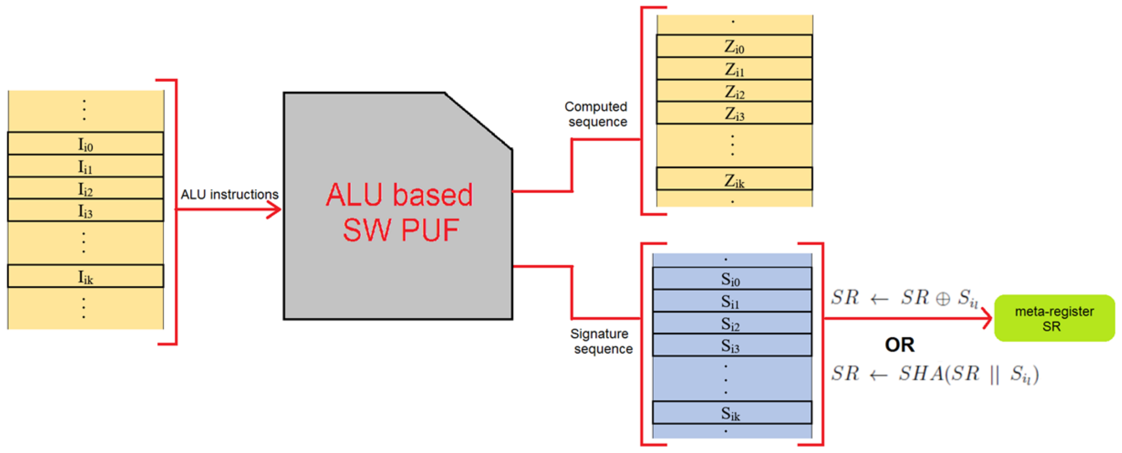


Figure 2.3 An Example of Software integrity measurement

2.7 Performance comparison of SW-PUF and TPM

Trusted Platform Module (TPM) is embedded within a trusted computing platform to provide a hardware root of trust. The TPM's design relies on the host platform for software measurements. It receives the measurements of the platform components performed by measuring software started on the CPU at boot stage. It stores these values in Platform Configuration Registers (PCRs) extending these values at each stage in a manner similar to our hash chaining. Thus, the TPM serves as a root of trust for reporting the measurement rather than performing the actual measurement. The TPM is able to hash and report the actual measurements from an untrusted environment. It is impossible for the TPM to verify these measurements. The Software PUF can solve this weakness in the TPM since all the measurements occur within the trust boundaries inside the CPU. Therefore, the SW-PUF by itself can provide a more robust TPM like static root of trust for endorsement. However, the SW-PUFs signature (hash values of the program instructions measured) is still a very aggregated measurement for the entire program. Verifying this kind of signature has two shortcomings: first the verifier needs to recompute everything which requires duplication of the entire effort, and second, the verifier needs to use the same SW-PUF at the same node since the SW-PUF is not reproducible at another node.

In order to establish a secure boot using SW-PUF, the computer system must first boot into a trusted state. A bootloader is typically the first program started after power-up or reset. By running the bootloader in a CPU with SW-PUF, a unique stamp or signature can be generated. Similar to the TPM secure boot, this stamp associated with the same hashing algorithm as in PCR of SW-PUF measurements could provide a strong root of trust.

One of the main security services offered by a TPM is remote attestation. This allows the system to verify its hardware and software configuration integrity to a remote party. Using SW-PUF, a stronger remote attestation protocol could be obtained. Figure 2.4 illustrates the basic steps on how such a system could work; additional modifications could be added such as using a nonce to ensure that old communications cannot be reused. This protocol is introduced only as a proof of concept.

First, at the manufacturing stage, each CPU is calibrated with a unique program (authentication program) to generate a SW-PUF signature. This specific authentication program and its signature need to be stored at a trusted third party called a Privacy CA. When a device receives a request for attestation, the attestor generates an Attestation Identity Key (AIK) and sends the public part of the key to the CA. The CA will then send back the authentication program to the attestor.

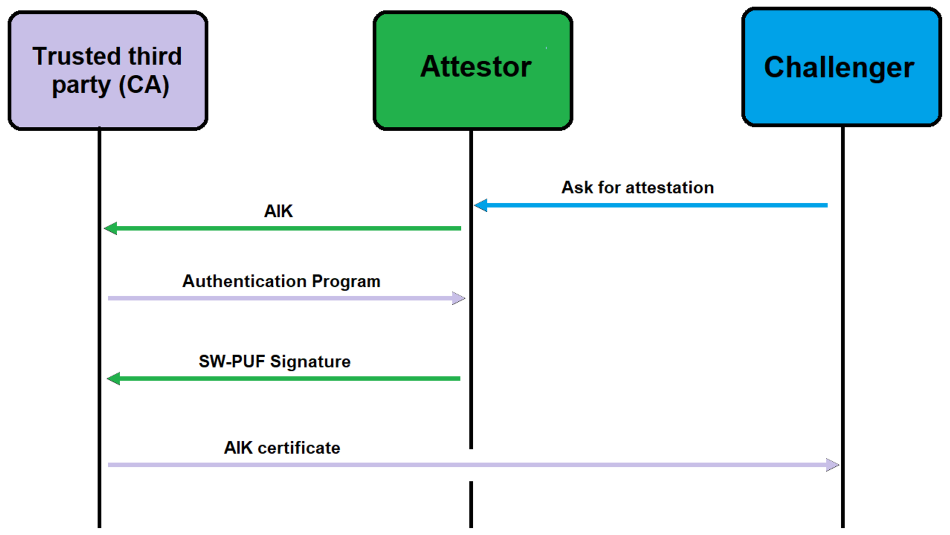


Figure 2.4 Remote attestation protocol using SW-PUF

After running the program, the attestor sends the SW-PUF signature to the third party. The CA will then generate a signed certificate and send it back to the attestor after validating the SW-PUF signature or PUF stamp. The attestor can now send the received AIK certificate to the challenger. Root of Trust for Storage (RTS) is another basic security service provided by a TPM. A key component of secure storage is the TPM-Sealing function. TPM-Sealing is a process to bind the encryption with device state; by which an encrypted (Sealed) data can be decrypted (Unsealed) only in the same TPM and at the same PCR state. As shown in figures 2.5 and 2.6 SW-PUF in forward and reverse modes could be used to seal/Unseal the data with the device state. The sealing process takes data and cryptographic key to generate SW-PUF signature and the encrypted value. They could then be added together to produce a sealed data package. To decrypt this package, the same CPU needs to run in reverse mode to reproduce the encrypted data. A typical decryption process can then be performed to generate the data.

As mentioned above, the SW-PUF can offered more advanced protection than the TPM.The following are some threat models where the SW-PUF performs better than a TPM.

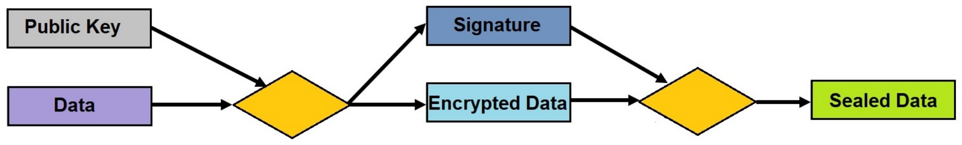


Figure 2.5 Sealing Data using Software-PUF

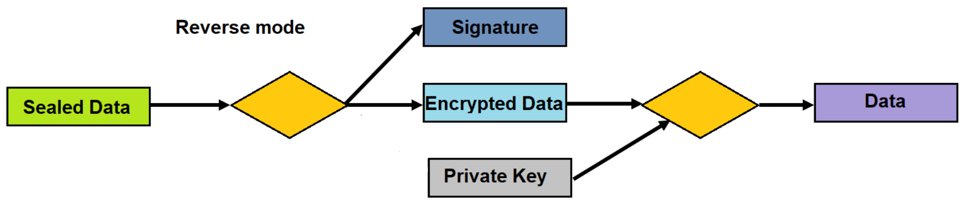


Figure 2.6 UnSealing Data using SW-PUF

Masquerade attack In remote attestation process, an attacker can send TPM measurements of another valid system to the third party. With a SW-PUF, instead, the signature of the CPU can be generated, available only using this particular CPU.

Malicious measurement agent A malicious agent may report incorrect integrity measurements to the TPM. However, all the measurements occur within the trust boundaries inside the CPU using the SW-PUF.

Hardware attacks An attacker may reset PCRs, and store new values in them. This attack can't be performed using the SW-PUF since there are no stored measurements.

2.8 Experimental and Simulation Results

2.8.1 Simulation Results

Simulations of 32-bit SW-PUF were performed in HSPICE k-2015.06 using predictive technology model HiSIM241 [13] for all NMOS/PMOS transistors with a V_{DD} of 3 V. The HiSIM241 model supports Design for Manufacturability (DFM). The value of a NSUBCDFM parameter is used to specify the substrate impurity concentration that will affect both the mobility and the threshold voltage. To simulate fabrication process variation, different values of NSUBCDFM (between 1.0e16 to 1.0e19) were assigned to each transistor in our design for each simulation. Matlab was used to generate random values for the NSUBCDFM parameter for each transistor which creates 16 different patterns of PUF instances. To evaluate our proposed PUF we studied three metrics: uniqueness, randomness, and reliability.

2.8.1.1 Uniqueness

Uniqueness measures the capability to distinguish between different devices. Hamming Distances (HD) between PUF responses are used to measure uniqueness. An ideal HD between any two PUF responses is 50% (16-bit). To evaluate the uniqueness of the proposed PUF, we ran simulations for each SW-PUF instance by applying identical challenge to produce a unique response for each. We then measured the HD between each pair of different responses (inter-chip). A total of 120 statistical data elements were found this way.

To determine the number of inverters or the early sampling delay needed to generate the best response for the SW-PUF in terms of randomness and uniqueness, we studied the response of our circuit for various inverter chain lengths used to generate the CLK signal time shift. Using Hspice for simulation allowed us to detect the output of the adder at any time without the need for an inverter chain. We evaluated the output data from the adder at 20%, 30%, 35%, and 40% of the average delay. Selection of these sampling intervals was to ensure that the HD between the early sampled adder outputs and the final result is greater than 8.

Figure 2.7 shows the Inter-chip HD distribution between pairs of 16 SW-PUF responses. The ALU outputs were sampled at 20%, 30%, 35%, and 40% of the average delay respectively. With 20% sampling period, the maximum Hamming distance was 11 (34% of PUF response bits); the minimum was 3 (1% of PUF response bits); and the average was 7.1 (21% of PUF response bits).

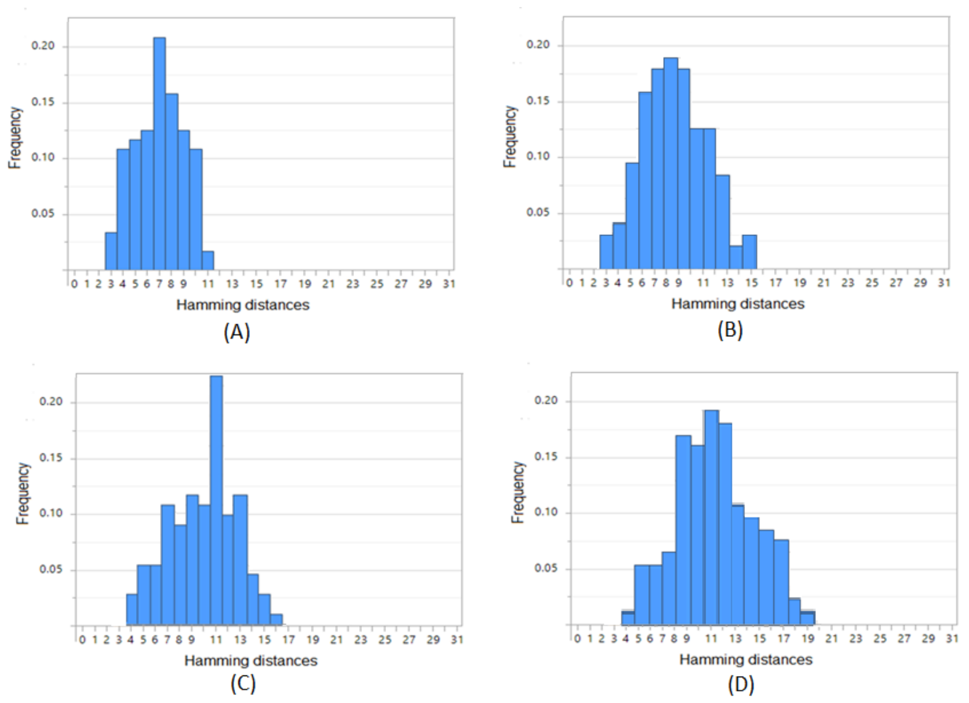


Figure 2.7 Inter-chip HD results for different sampling delay = (A) 20% , (B) 30%, (C) 35%, and (D) 40% respectively

While with 40% sampling period, the maximum HD was 17 (53% of PUF response bits); the minimum was 4 (13% of PUF response bits); and the average was 10.2 (32% of PUF response bits).

To evaluate the uniqueness of the SW-PUF on the same ALU under different data inputs (Intra-chip), we measured the average HD distribution between a pair of output data on the same device PUF instance with different set of input data. In this case, as shown in Figure 2.8, the maximum HD between any pair of PUF responses was 25 (78% of PUF response bits); the minimum was 10 (31% of PUF response bits); and the average was 15.8 (50% of PUF response bits).

2.8.1.2 Randomness

Randomness evaluates two properties in a PUF signature by analyzing the distribution of 0's and 1's, the independence and the uniformity. In this paper we have used the standard statistical test suite of the National Institute of Standard and Technology (NIST) presented by Bassham

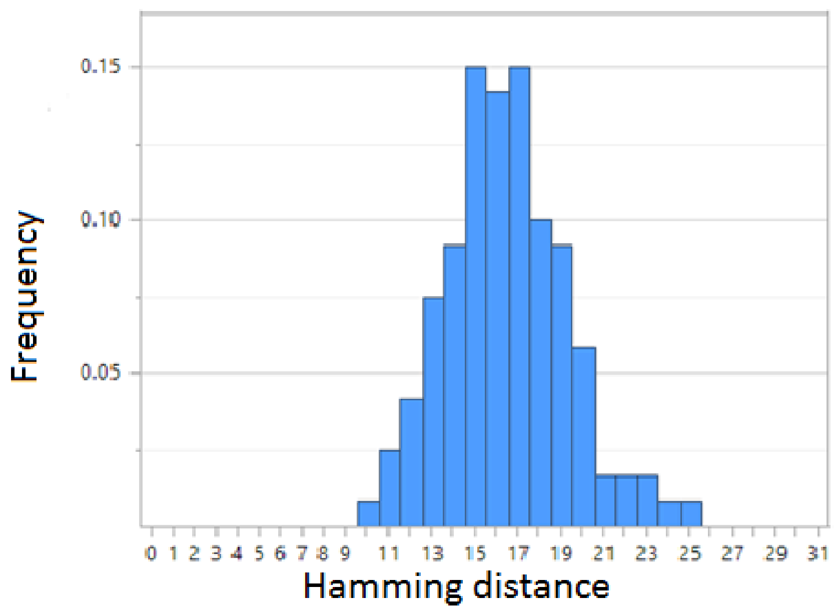


Figure 2.8 Intra-chip HD results under inputs variations and fixed sampling delay

et al. (2010) to evaluate the responses of the SW-PUF and the reversible SW-PUF. NIST test suite is commonly used for empirical statistical testing of uniform random number generators. The minimum pass rate for each statistical test is approximately 90.6%. We have applied the NIST tests to 512-bit stream that was produced from the 16 PUF instances. Table 2.1 shows the statistical tests results that were observed at 20% (second column) and 40% (third column) sampling periods for the SW-PUF. It was found that only one category (rank), out of fifteen statistical tests applied, failed - in the case where the outputs of the SW-PUF were sampled at 20% of the average delay. However, two categories (rank and non-overlapping template) failed with the 40% sampling period. This shows that a tradeoff between randomness and uniqueness can be achieved. Larger delay provides more uniqueness but less randomness.

2.8.1.3 Reliability

Reliability measures robustness of a PUF in the presence of environmental variations. Temperature variations are the main factor that affect the stability of a PUF response. Therefore,

Table 2.1 Summary results of statistical tests on a SW-PUF response observed at sampling period of 20% and 40% of the average delay

#Statistical Test	Avg. Success Ratio at 20%	Avg. Success Ratio at 40%
Frequency	96.875%	96.875%
Block Frequency	93.75%	93.75%
Cumulative Sums	96.875%	96.875%
Run	96.875%	96.875%
Longest Run	96.875%	100%
Rank	0%	0%
Discrete Fourier Transform	100%	96.875%
NonOverlappingTemplate	95.98%	89.531%
OverlappingTemplate	100%	100%
Universal	100%	100%
RandomExcursionsVariant	100%	100%
Approximate Entropy	100%	100%
Serial	93.75%	95.31%
LinearComplexity	100%	96.875%
the minimum pass rate for each statistical test is approximately 90.6%		

reliability can be determined by varying the operating temperature and comparing the responses with a reference response obtained under nominal temperature conditions for the same input. We used the following equation for calculating the reliability of SW-PUF responses:

$$Reliability = \left(1 - \frac{1}{K}\sum_{i=0}^{n-1}\frac{HD(R_i, R_{i,j})}{n}\right) \times 100\% \tag{2.3}$$

where R_i is the reference response at room temperature 25°C, $R_{i,j}$ is a response observed under varying temperature value for the same challenge, n is the number of bits of SW-PUF, and K is the number of samples. Figure 2.9 shows the average reliability of the sixteen PUF instances with operating temperature from -25°C to 75°C for four cases: sampling periods at 20%, 30%, 35%, and 40% of the average delay. The worst reliability is 91% and occurs at -25°C. However, the SW-PUF has acceptable stability with more than 96% reliability for temperatures from -10°C to 65°C.

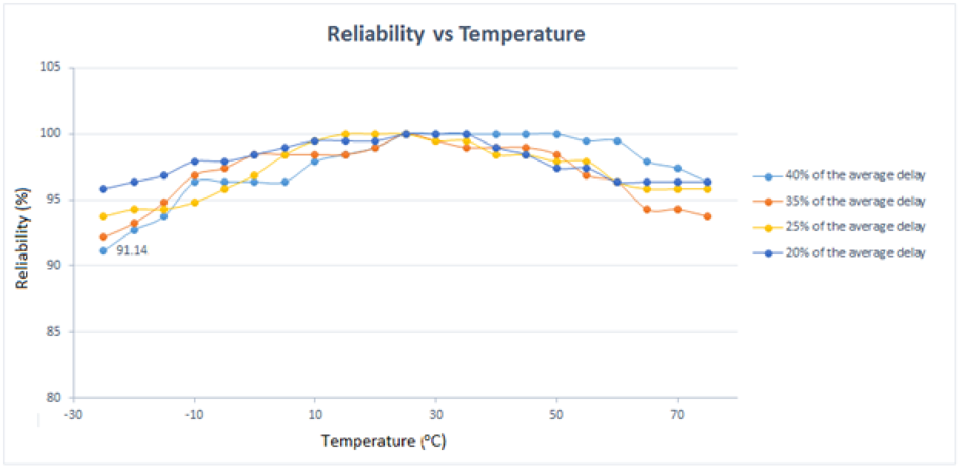


Figure 2.9 The reliability of SW-PUF against temperature variations

2.8.2 Experimental Results with FPGA Implementations

In order to broaden the applicability of our results, we synthesized our SW-PUFs on an FPGA platform as well. The Field Programmable Gate Array (FPGA). PUFs results can be compared with the results obtained from the Hspice simulation. We implemented a 64-bit SW-PUF on an Altera DE2 Development and Education Board. Since we needed to evaluate fabrication variation between pairs of different FPGAs, we implemented our design on eight identical FPGAs.

Figure 2.11 shows the HD distributions between a pair of 8 SW-PUFs responses from different FPGA devices with identical inputs and 2 inverter delay sampling period. With 2 inverter delay sampling period, the maximum HD was 25; the minimum was 3; and the average was 13.2. This is worse than the uniqueness profile of HSPICE based simulations. However, better uniqueness can be achieved by improving the routing for the SW-PUF.

As we did in Hspice experiments, to evaluate the uniqueness of the SW-PUF based on the adder input data, we measured the average HD distribution between pairs of the output responses on the same FPGA with different set of input data(Intra-FPGA) . As shown in Figure 2.10, the maximum HD between any pair of the PUF response was 38 (59% of PUF response bits); the minimum was

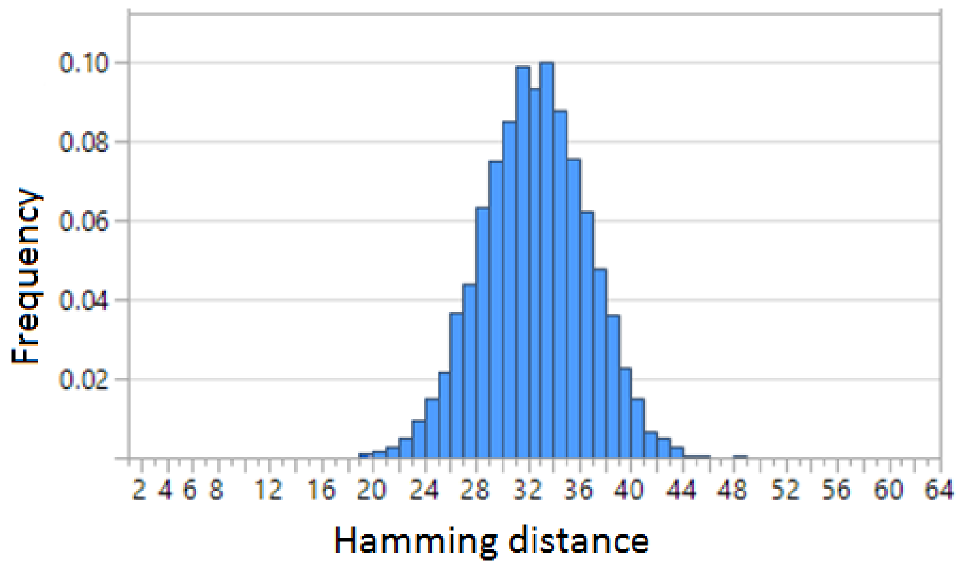


Figure 2.10 Intra-FPGA HD results under inputs variations

25 (39% of PUF response bits); and the average was 32.3 (50% of PUF response bits). This is similar to the SPICE based PUF responses.

2.8.3 Area and energy comparison of SW-PUF and TPM

The SW-PUF design has been developed using Verilog-HDL and synthesized in Cadence RTL compiler using 45nm CMOS technology. In earlier work presented by Hamadeh et al. (2017), a similar design flow is pursued to compare a TPM and distributed TPM. In that work, a Residue Number System (RNS) based homomorphic share scheme for TPM is used among several IoT devices within a cluster. The TPM functionality is distributed to 4 or 10 slices in order to reduce the cost of the communication between the devices in the same cluster. While A full TPM functionality is required at the inter-cluster level, where each operation is divided between the cluster member. A VLSI area and energy for a subset of TPM commands were computed for TPM with 1024-bit RSA key size (full functionality), 256-bit RSA key size (4 slices), and 160-bit RSA key size (10

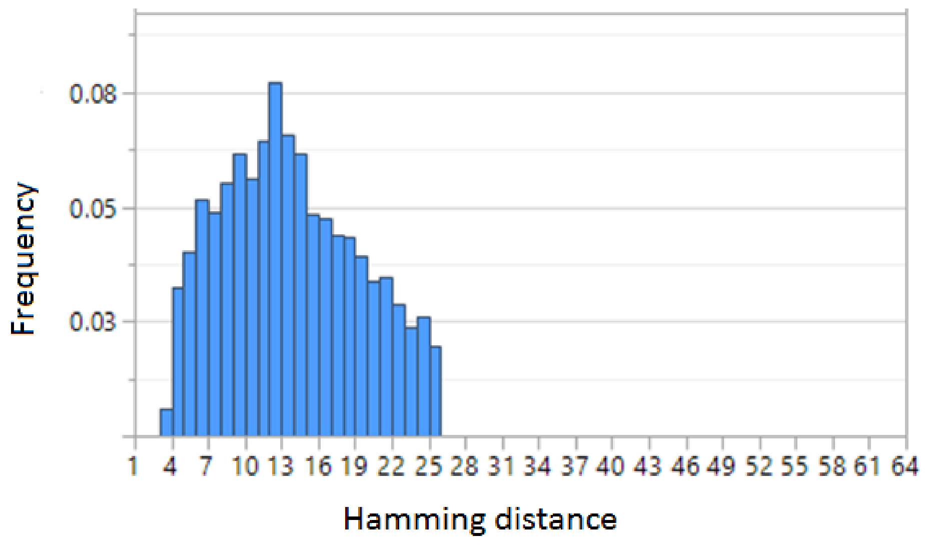


Figure 2.11 Inter-FPGA HD results

slices). As shown in Table 2.2. The VLSI area and energy consumption of the proposed SW-PUF and Reversible SW-PUF are negligible compared to the smallest slice of the TPM which provides a more efficient design.

Table 2.2 Area and energy results of SW-PUF, Reversible SW-PUF and TPM. Note that TPM is evaluated for multiple key sizes to reflect various classes of IoTs.

	SW-PUF	Reversible SW-PUF	TPM(1024-RSA)	TPM(256-RSA)	TPM(160-RSA)
Total Area	0.029mm^2	0.789mm^2	618.76mm^2	600.52mm^2	598.25mm^2
Total Energy	0.00071 nJ	0.00715 nJ	4661.86 μJ	220.12μJ	25.56μJ

2.8.4 Program Level Hash Digest of SW PUF Responses

The individual program instruction level SW-PUF responses are hash-extended to generate a program level signature. We wanted to evaluate such signatures over a collection of programs and execution paths through the same program to assess if the program level signatures are able to differentiate different programs or execution paths. This experiment is a preliminary assessment of this capability. A cycle accurate CPU simulator like SimpleScalar Austin et al. (2002) can compute individual instruction inputs, but not the SW-PUF behavior. Pin tool Reddi et al. (2004) (dynamic binary instrumentation tool) was used to collect instructions and instruction input data values from SPEC 2006 CPU benchmarks executed on Intel x86 32-bit microprocessors. Since, the SW-PUF responses were computed with HSPICE, only a small number of such responses could be evaluated practically. Hence, a random 100 instructions for each benchmark were sampled from its execution trace. Their PUF stamp sequences were then simulated with HSPICE k-2015.06. The program level signatures were generated by XOR'ing the 100 randomly sampled instruction level signatures. Figure 2.12 shows the average HD between the program level PUF stamps over 16 chips across the SPEC 2006 benchmarks. As shown in Figure 2.12, the minimum HD between a pair of instruction level signatures was 20.9%. Any hash function based signature extension will result in 50% HD between a pair of program stamp digests.

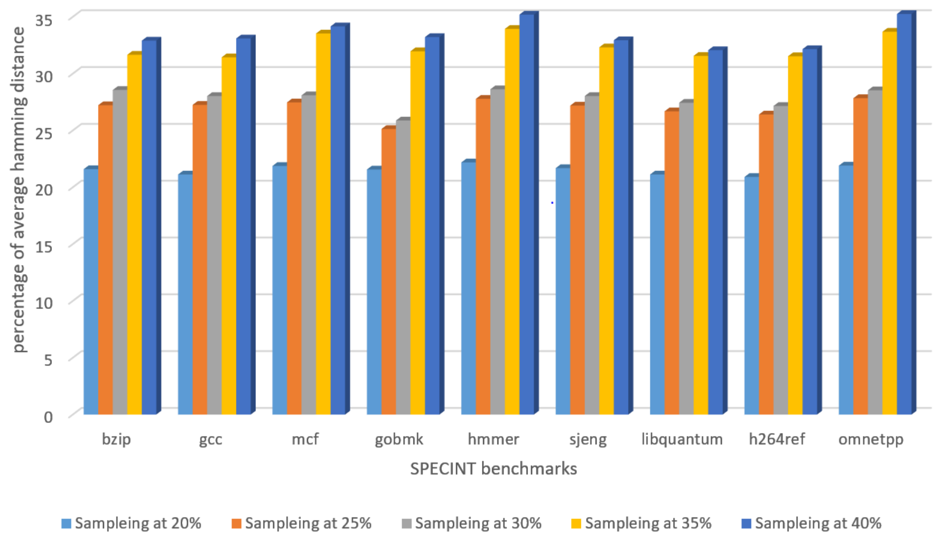


Figure 2.12 HD of Program Level Hash-Extended Signatures for different devices.

2.9 Conclusions

In this paper, we present a novel PUF design that is suitable for Software Protection & Trusted Computing Base. We show that the proposed SW-PUF is better suited for software protection than a TPM. We also demonstrate that it can provide a more robust static root of trust. The SW-PUF design has been implemented on an FPGA platform (Altera DE2 Development and Education Board). In addition, a simulation analysis in HSPICEk-2015.06 using predictive technology model HiSIM241 was performed. The responses measured from the SW-PUF have a uniqueness of about 30%, excellent randomness, and a reliability of 96% at temperatures from -10°C to 65°C. Wang et al. (2018) generate a stable response of a PUF by utilizing random hard defect generated from Directed Self Assembly (DSA) process. A similar mechanism could be used to generate an ideal response for the proposed SW-PUF.

2.10 References

Austin, T., Larson, E., and Ernst, D. (2002). Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67.

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., Heckert, N. A., Dray, J. F., and Vo, S. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Gaithersburg, MD, United States.

Burks, A. W., Goldstine, H. H., and von Neumann, J. (1987). *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument (with Arthur W. Burks and Herman H. Goldstine*, volume 12 of *Charles Babbage Institute reprint series for the history of computing*. MIT Press, Cambridge, MA, USA.

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*.

Choi, P. and Kim, D. K. (2012). Design of security enhanced tpm chip against invasive physical attacks. In *2012 IEEE International Symposium on Circuits and Systems*, pages 1787–1790. IEEE.

Devadas, S., Suh, E., Paral, S., Sowell, R., Ziola, T., and Khandelwal, V. (2008). Design and implementation of puf-based "unclonable" rfid ics for anti-counterfeiting and security applications. In *2008 IEEE International Conference on RFID*, pages 58–64.

Garside, J. D. (1993). A cmos vlsi implementation of an asynchronous alu. In *Proceedings of the IFIP WG10.5 Working Conference on Asynchronous Design Methodologies*, pages 181–192, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

Guajardo, J., Kumar, S. S., Schrijen, G.-J., and Tuyls, P. (2007a). Fpga intrinsic pufs and their use for ip protection. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 63–80, Berlin, Heidelberg. Springer-Verlag.

Guajardo, J., Kumar, S. S., Schrijen, G. J., and Tuyls, P. (2007b). Physical unclonable functions, fpgas and public-key crypto for IP protection. In *FPL*, pages 189–195. IEEE.

Hamadeh, H., Chaudhuri, S., and Tyagi, A. (2017). Area, energy, and time assessment for a distributed tpm for distributed trust in iot clusters. *Integration*, 58:267–273.

Herder, C., Ren, L., van Dijk, M., Yu, M.-D., and Devadas, S. (2017). Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Transactions on Dependable and Secure Computing*, 14(1):65–82.

Kawa, J., Chiang, C. C., and Camposano, R. (2006). EDA challenges in nano-scale technology. In *CICC*, pages 845–851. IEEE.

Kong, J., Koushanfar, F., Pendyala, P. K., Sadeghi, A. R., and Wachsmann, C. (2014). Pufatt: Embedded platform attestation based on novel processor-based pufs. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6.

Lebedev, I., Hogan, K., and Devadas, S. (2018). Secure boot and remote attestation in the sanctum processor. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 46–60. IEEE.

Li, D., Guo, H., and Xu, J. (2016). Enhancing tpm security by integrating sram pufs technology. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 82–93. ACM.

Müller, K.-U., Ulrich, R., Stanitzki, A., and Kokozinski, R. (2018). Enabling secure boot functionality by using physical unclonable functions. In *2018 14th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, pages 81–84. IEEE.

Nithyanand, R. and Solis, J. (2012a). A theoretical analysis: Physical unclonable functions and the software protection problem. In *2012 IEEE Symposium on Security and Privacy Workshops(SPW)*, volume 00, pages 1–11.

Nithyanand, R. and Solis, J. (2012b). A theoretical analysis: Physical unclonable functions and the software protection problem. In *IEEE Symposium on Security and Privacy Workshops*, pages 1–11. IEEE Computer Society.

Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

Parhami, B. (2009). *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Inc., New York, NY, USA, 2nd edition.

Reddi, V. J., Settle, A., Connors, D. A., and Cohn, R. S. (2004). Pin: A binary instrumentation tool for computer architecture research and education. In *Proceedings of the 2004 Workshop on Computer Architecture Education: Held in Conjunction with the 31st International Symposium on Computer Architecture*, WCAE '04, New York, NY, USA. ACM.

Scheel, R. A. and Tyagi, A. (2015). Characterizing composite user-device touchscreen physical unclonable functions (pufs) for mobile device authentication. In *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*, TrustED '15, pages 3–13, New York, NY, USA. ACM.

Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA. ACM.

Wang, W.-C., Yona, Y., Diggavi, S. N., and Gupta, P. (2018). Design and analysis of stability-guaranteed pufs. *IEEE Transactions on Information Forensics and Security*, 13(4):978–992.

CHAPTER 3. PROBABILISTIC VERIFICATION OF OUTSOURCED COMPUTATION BASED ON NOVEL REVERSIBLE PUFS

Modified from a paper accepted by *ESOCC-2020 conference*

Hala Hamadeh, Abdallah Almomani, and Akhilesh Tyagi

3.1 Abstract

With the growing number of commercial cloud-computing services, there is a corresponding need to verify that such computations were performed correctly. In other words, after a weak client outsources computations to an untrusted cloud, it must be able to ensure the correctness of the results with less work than re-performing the computations. This is referred to as verifiable computation. In this paper we present a new probabilistic verifiable computation method based on a novel Reversible Physically Unclonable Function (PUF) and a binomial Bayesian Inference model. Our scheme links the outsourced software with the cloud-node hardware to provide a proof of the computational integrity and the resultant correctness of the results with high probability. The proposed Reversible SW-PUF is a two-way function capable of computing partial inputs given its outputs. Given the random output signature of a specific instruction in a specific basic block of the program, only the computing platform that originally computed the instruction can accurately regenerate the inputs of the instruction correct within a certain number of bits. To explore the feasibility of the proposed design, the Reversible SW-PUF was implemented in HSPICE using 45 nm technology. The probabilistic verifiable computation scheme was implemented in C++, and the Bayesian Inference model was utilized to estimate the probability of correctness of the results returned from the cloud service.Our proof-of-concept implementation of Reversible SW-PUF exhibits good uniqueness compared to other types of PUFs, and exhibits 100% reliability for temperatures between-10°C to 65°C. The randomness was measured with the NIST suite of 14

tests wherein 12 tests were consistently passed. Finally, we demonstrate our verifiable computation approach on a matrix computation. We show that it enables faster verification than existing verification techniques.

3.2 Introduction

Verifiable Computations (VC) have attracted enormous interest and attention with the recent growth in cloud computing. The concept of verifiable computation allows a lower-resource client to outsource the computation of a program to an untrusted cloud. With a proof provided by the cloud, the client can verify that the results produced are consistent with the program specification and that the computations were performed correctly. To be viable, the effort of performing the verification must be negligible compared to the actual computation. In recent years, interest in physically-unclonable functions (PUFs) has evolved. PUFs have been deployed in different applications because of their ability to generate "digital fingerprints" of unique identities for a physical system. SW-PUF Hamadeh and Tyagi (2019) is a specific type of PUF that binds software execution to the exact hardware platform and produces unique signatures at various points in the software's execution. The SW-PUF signature is a promising candidate for providing a proof that a specific computation was performed on a specific platform. By expanding the capabilities of a SW-PUF to include invertibility and commutativity, we achieve elements of verifiable computation. Invertibility is achieved by capturing a physical attribute such as time when an output bit settles using reversible functions. Reversibility is obtained with transmission gates.

Contributions:

The Reversible SW-PUF: We present a novel reversible PUF design based on transmission gates capable of generating partial input word, (challenge) consistent with a unique hardware-specific output (response) for a given computing platform.

Probabilistic Verification Computation based on Reversible SW-PUF Scheme: We develop a probabilistic randomized verification method built on top of the Reversible SW-PUF, in which the server (cloud computing node) computes the results from a program and generates

SW-PUF signatures as proof of performing the computation correctly. For verification of the results, the client can randomly pick a small number of PUF signature pairs (input, output). It can then send only the response part to the server. Server can compute the corresponding challenge by running its reversible SW-PUF in reverse. The client can then confirm whether the challenge for each verification point indeed matches the original signature/proof. Finally, the client can use a Bayesian Inference Dempster (1968) model to obtain probabilistic proof of the server's results.

Implementation and Evaluation: The Reversible SW-PUF has been evaluated in HSPICEk-2015.06 using predictive technology model HiSIM24. We demonstrate the efficiency and effectiveness of the Reversible SW-PUF. The invertibility of the Reversible SW-PUF offers an alternate, possibly more efficient, way of providing a proof of verifiable computation. To evaluate the performance of the probabilistic verification computation approach, we implemented a case study wherein a verification of matrix multiplication was performed and compared with previous works. The results show increased efficiency.

The rest of the paper is organized as follows. In Section 3.3, we summarize the existing work on PUFs and describe relevant concepts. Section 3.4 presents reversible SW-PUF which is the building block for verifiable computation. In section 3.5, we propose the Verifiable Computation Scheme. The implementation and evaluation of the Reversible SW-PUF is presented in Section 3.6. To evaluate our approach, we presents a case study of matrix multiplication in section 3.7. Finally, Section 3.9 concludes the paper.

3.3 Related Work

Physical Unclonable Functions (PUFs) were first proposed in 2002 by Pappu et al. (2002b). Most popular PUFs are implemented based on timing and delay variation of silicon circuits. Our proposed design approach is also based on such timing parameters variability. On the other side, there is a new trending to replace all digital logic with the reversible logic; therefore, There are several research efforts to build reversible logic to replace the classical logic. For example, Dey et al. (2019) proposed a new design of reversible PUF based on the Feynman gate.

PUFs are suitable for use in many application to provide hardware security such as the work presented by Choi and Kim (2012), Li et al. (2016), Lebedev et al. (2018) and Müller et al. (2018). This paper focus on applying PUF in the verifiable computation application. Three main solutions were proposed to support verifiable computation in the literature: verifiable computation based on Trusted Computing Sailer et al. (2004), Chen et al. (2006). The main drawback of this approach was the assumption that physical protections cannot be defeated. A second method, verifiable computation with a Non-Interactive Argument, is described in Parno et al. (2013), Parno et al. (2016). This approach is not practical because it relies on complex Probabilistically Checkable Proofs (PCPs) or fully-homomorphic encryption (FHE). Finally, verifiable computation with Interactive Proofs Vu et al. (2013), Thaler et al. (2012) has been proposed. While this approach is often efficient, it applies to only a narrow class of computations.

3.4 The Reversible SW-PUF

The design of the Reversible SW-PUF is an extension of our previous work on the SW-PUF Hamadeh and Tyagi (2019). As in the original SW-PUF, the ALU signatures of an instruction on the reversible software PUF are generated from an early sampling of the ALU results. However, in the reverse mode, the roles of inputs and outputs are reversed, and the early sampling is done on the original input end. Reversible SW-PUF has two modes: forward and reverse. The forward mode is similar to the SW-PUF where it generates a unique signature by capturing the delay variations of carry propagation in ripple-carry adders (which is a basic component in an ALU). The delay variation is caused by instruction input values and the silicon fabrication foundry variations. The reverse mode computes the partial inputs from the signature and the instruction output. Early sampling captures a subset of original input bits correctly in a platform specific manner, which itself is a platform specific secret. Only the computing platform that originally computed the instruction can regenerate the inputs of the instruction accurate within a certain number of bits. For simplicity, we illustrate the Reversible SW-PUF design by an example of a 4-bit adder (Figure 3.1). In this design, Reversible SW-PUF is implemented in reversible logic Toffoli (1980), Bennett (1973).

Fredkin gate Fredkin and Toffoli (1982) is used as the Boolean basis for conservative logic because it is universal.

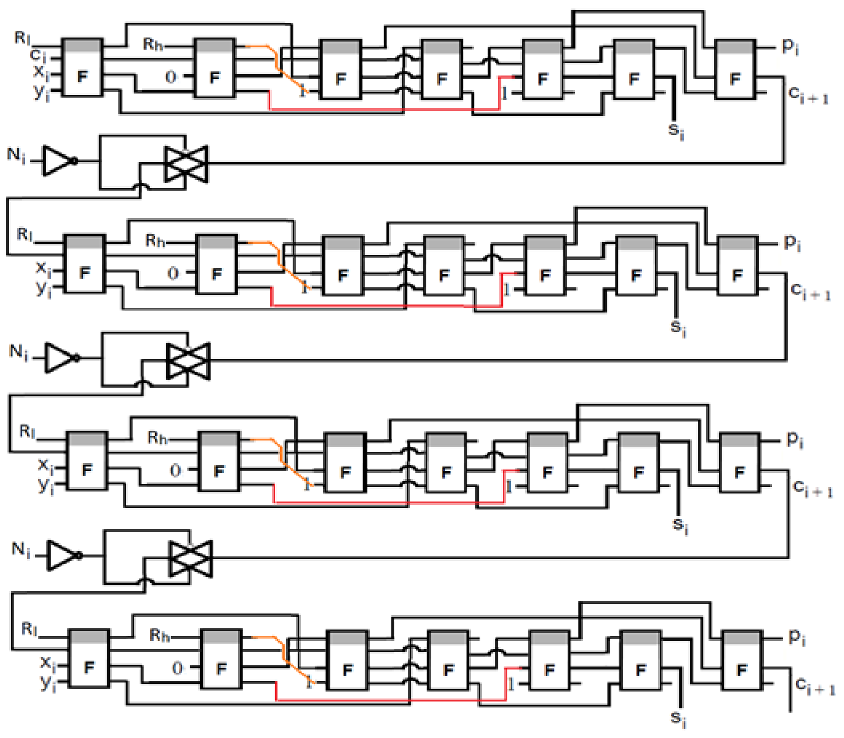


Figure 3.1 Reversible SW-PUF Design (4-bit responses)

Since Fredkin gates are based on Transmutation Gates(TGs), and TGs are slow compare to a regular gate, we propose to use two ALUs (fast-ALU, rev-ALU). The actual computation values consumed by the following program instructions occur at the fast-ALU. The rev-ALU is used only for verification. We need to ensure that the execution time of the rev-ALU both in the forward and the backward computation modes is less than the execution time of the fast-ALU. For an N-bit fast-ALU, the rev-ALU is divided into several segments of $k < N$ bits. To define the maximum length of each segment, the following constraint is used:

$$\left(\frac{\texttt{Tfast\_ALU}}{\texttt{TB\_rev\_ALU} + \texttt{TF\_rev\_ALU}}\right) \geq 1 \tag{3.1}$$

Where Tfast_ALU is the worst-case delay to execute an instruction using the fast ALU. TB_rev_ALU and TF_rev_ALU are the worst case delays over all segments in forward and backward computation using revALU. As shown in Figure 3.2, we propose to divide the reversible ALU into four segments of 8-bits each for a 32-bit fast-ALU. This satisfies the delay constraint equation and still provides an acceptable uniqueness.

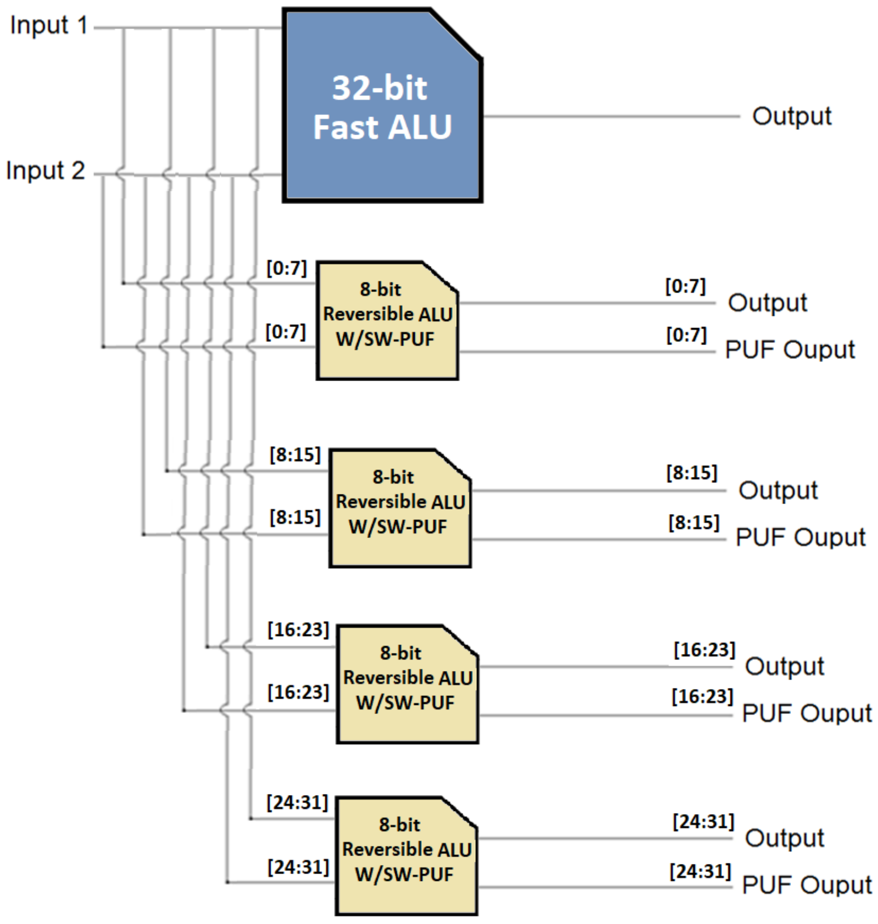


Figure 3.2 ALU based Reversible SW-PUF

3.5 Verifiable Computation Scheme

In this section an efficient Verifiable Computation Scheme based on Reversible SW-PUF is proposed. The proposed scheme fits with a probabilistic consistency guarantee. In this scheme, we are interested in estimating the probability of a cloud service to return a correct result for the outsourced function. The main idea is to bind the verification scheme to the cloud service hardware by entangling the computation with the SW-PUF. When the cloud computes the function, an instruction sequence $I0$, $I1$, $I2$, $\ldots$, In for each instruction I_l generates relevant attributes which are the two data inputs for the lth instruction - X_0^l, X_1^l, the instruction output Y^l, and the PUF output P_l.

Effectively, the cloud node generates a signature (response) for each instruction (challenge) in the execution path. This entire sequence of challenge-response pairs will be returned to the client as a proof of computational consistency.

For the verification process, the client can verify the behavior of a program slice of variable granularities. Most straightforward granularity is to verify an individual instruction behavior. Pick a random challenge-response(C_k, R_k) pair of an instruction I_k to verify. The client needs to send the response part (the instruction output Y^k, and the PUF output P_k) to the cloud node. The cloud instantiates the reversible SW-PUF to re-compute the challenge from the response (the data inputs of the instruction $(X_0'^k, X_1'^k)$. Only the cloud node that computed the original signature will be able to compute the inverse PUF, so that $(X_0'^k, X_1'^k)$ is consistent with the (X_0^k, X_1^k) in the original computation's proof of consistency within a large number of bits. We assume that over all the clients and programs, the amount of data is too large to be archived by the cloud node preventing a look-up based response to the verification step.

Repeating this verification process for all the n instructions is not feasible because a large number of instructions could be executed during a program run. As we discuss later, an alternative approach to pick a subset of instructions is used to increase the confidence interval for the verification while maintaining an efficient verification.

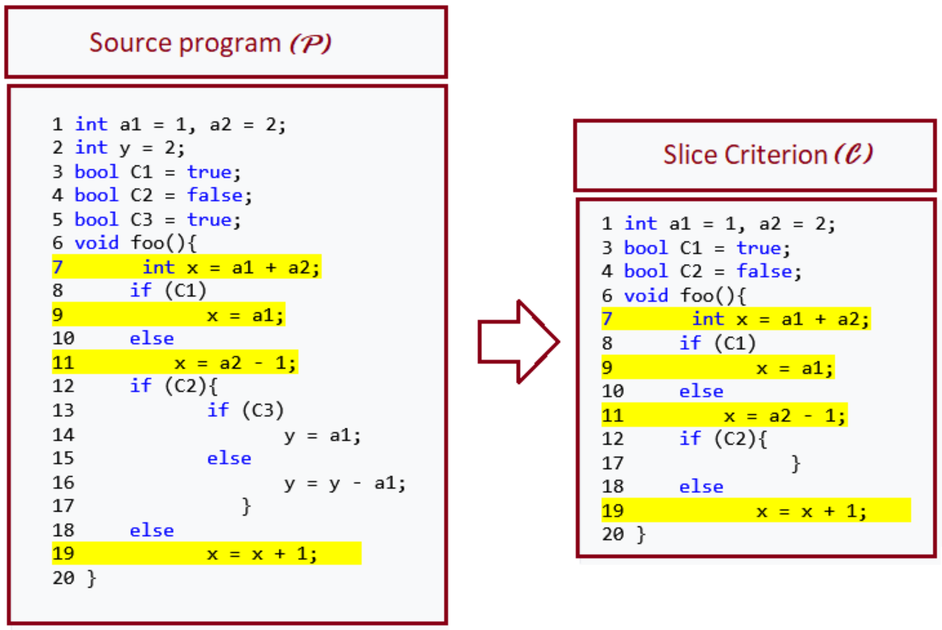
Static program slices raise this granularity naturally. Static program slicing was introduced by Weiser Weiser (1981). It is a technique for reducing a program to a minimal form that still retains the original program computation for a given variable at a chosen point. An example of a program and a backward slice with respect to slicing criterion < 19, x $>$ is given in Figure 3.3 and 3.4. Merging the program slicing technique with our verification scheme leads to a more efficient Verifiable Computation. A program slice's input/output consistency can be established with the Reversible SW-PUF method. For a program slice, all of the instructions in its execution flow can be verified leading to a deterministic verification. The program slices can be extracted to maximize certain static properties.

Figure 3.5 describes an example of a client that wants to run the program on a cloud server using the proposed protocol:

Client:
- Send the program P and the inputs to the server.
- Generate a backward slice S using control flow graph shown in Figure 3.4.
- Run slice S on a trusted device.
- Collect the inputs (challenges) of the Reversible SW-PUF (forward mode) for each instruction in the execution path for the slice.

Server:
- Run the program and generate SW-PUF signatures (the responses R_i in challenge-response pairs) for each instruction in the execution path of the program P.
- Send the results of the program in addition to the SW-PUF signatures sequence R_i to the client.

Client:
- Perform a consistency check on the SW-PUF inputs (challenges C_i) generated for the slice S, and the PUF signatures R_i received form the server.
- Pick a subset of the SW-PUF signatures $R_{i_0}, R_{i_1}, \ldots, R_{i_k}$ to verify. More comprehensive verification will verify each instruction within the slice S. An interactive verification can pick a subset of instructions from the dominant control-flow paths within S. Regardless, only the response of an individual instruction R_{i_l} is sent to the server at a time.



```
1 int a1 = 1, a2 = 2;
2 int y = 2;
3 bool C1 = true;
4 bool C2 = false;
5 bool C3 = true;
6 void foo(){
7         int x = a1 + a2;
8       if (C1)
9               x = a1;
10      else
11              x = a2 - 1;
12      if (C2){
13              if (C3)
14                      y = a1;
15              else
16                      y = y - a1;
17                 }
18      else
19              x = x + 1;
20 }
```

```
1 int a1 = 1, a2 = 2;
3 bool C1 = true;
4 bool C2 = false;
6 void foo(){
7         int x = a1 + a2;
8       if (C1)
9               x = a1;
10      else
11              x = a2 - 1;
12      if (C2){
17                 }
18      else
19              x = x + 1;
20 }
```

Figure 3.3 An Example of Program slicing

Server: • Run the SW-PUF in reverse mode with its output asserted to the response R_i to be verified. The reverse computation generates the input (challenge C_i) of the PUF.

Client: • Confirm that the challenge C_i does match the response R_i for this SW-PUF given the (C_i, R_i) pairs from the original program execution at the server. As we describe later, a reversible SW-PUF match means that the challenge C_i does not equal the inputs of the ALU at specific bit positions given an early sampling period.

- Iterate over the slice S instructions, and over all the slices.

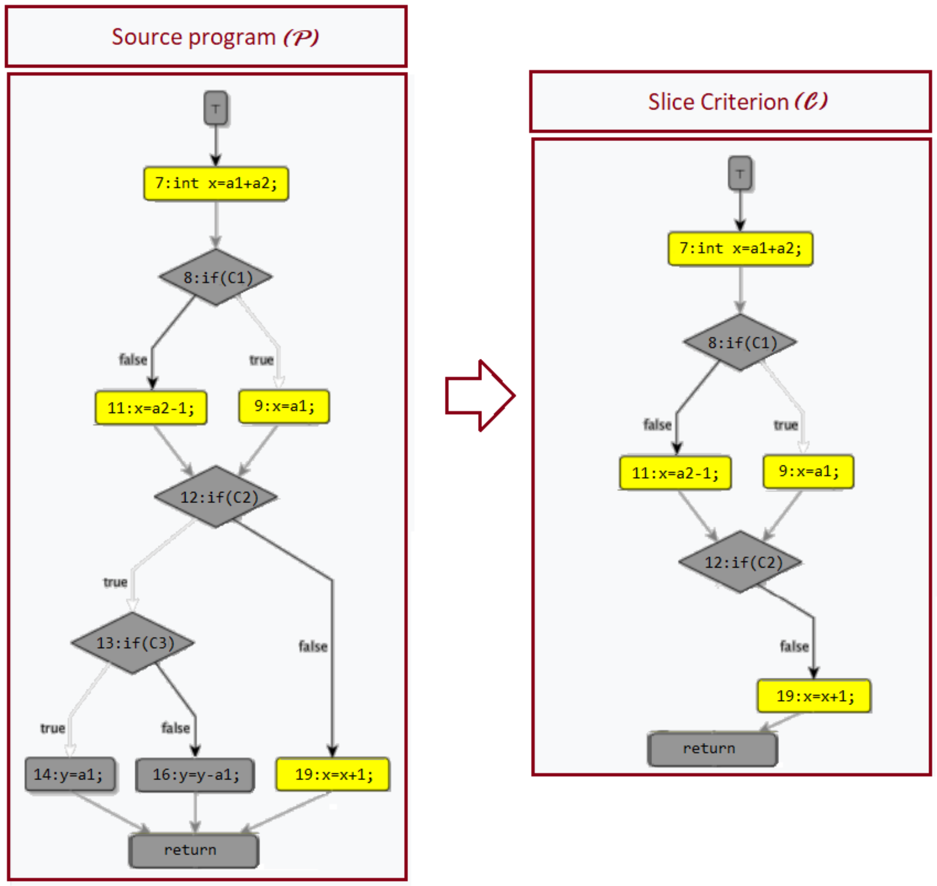


Figure 3.4 An Example of control flow graph

- The slices to be verified will ideally be picked so that the probability of behavior misrepresentation for the adversary are minimized.
- Apply Bayesian Inference model to get probabilistic proof about the server's results.

For choosing the slice set in our scheme, two elements are critical: the size of the slice, and the number of slices. Since small slices result in more efficient verification (According to Binkley et al. (2007), the average slice contains just under one-third of the program) we propose to use a selection method based on the super-node Zhang et al. (2018) algorithm to reduce the verification effort. However, certain types of program control flow graphs may not be amenable to small slices, and in

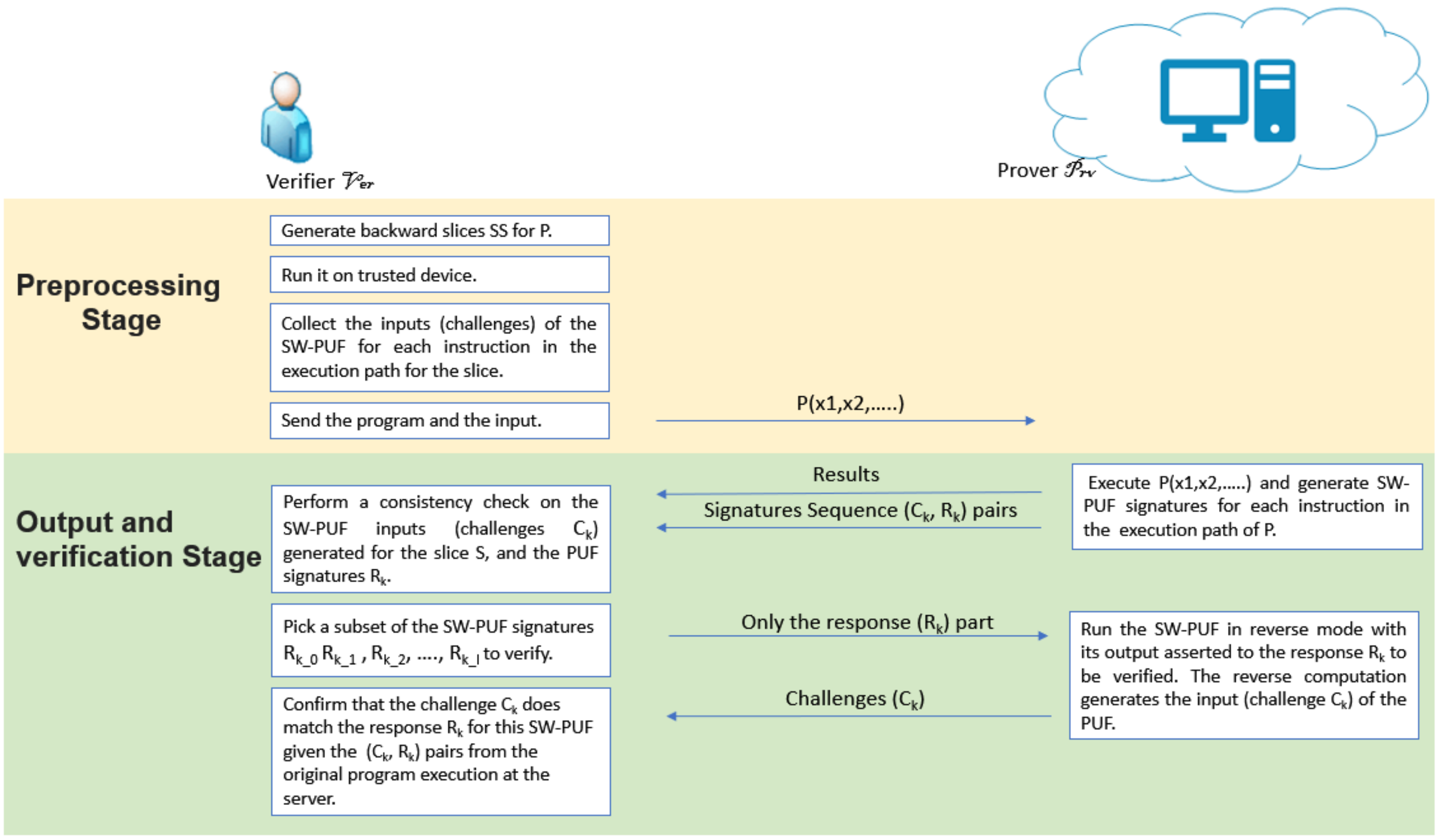

Verifier $\mathcal{V}_{er}$
Prover $\mathcal{P}_{rv}$
Preprocessing Stage
Generate backward slices SS for P.
Run it on trusted device.
Collect the inputs (challenges) of the SW-PUF for each instruction in the execution path for the slice.
Send the program and the input.
P(x1,x2,.....)
Output and verification Stage
Results
Signatures Sequence (C_k, R_k) pairs
Execute P(x1,x2,.....) and generate SW-PUF signatures for each instruction in the execution path of P.
Perform a consistency check on the SW-PUF inputs (challenges C_k) generated for the slice S, and the PUF signatures R_k.
Pick a subset of the SW-PUF signatures R_{k_0} R_{k_1} , R_{k_2}, ..., R_{k_l} to verify.
Only the response (R_k) part
Run the SW-PUF in reverse mode with its output asserted to the response R_k to be verified. The reverse computation generates the input (challenge C_k) of the PUF.
Challenges (C_k)
Confirm that the challenge C_k does match the response R_k for this SW-PUF given the (C_k, R_k) pairs from the original program execution at the server.

Figure 3.5 The proposed protocol

such a case, different methods could be applied. As in any interactive proof system, increasing the number of slices will increase confidence in the computed results. To provide a desired probabilistic proof about the server's results, we propose to use a Bayesian Inference Dempster (1968) model to determine the appropriate number of static slices required.

Slices Selection:

Given a program P that contains a set of instructions S, our goal is to find a subset of S called M such that M exhibits the same behavior as S with respect to one of the program outputs. Once we find M, while we want to generate static slices SS that go through M, the selection of M must be based in some randomized algorithm to prevent an adversary from producing the same M to cheat. For choosing M, we used the algorithm in Zhang et al. (2018), for selecting all the super-nodes in P as our set of desired nodes. A super-node is formed from a strict dominator-post-dominator pair. A node X is defined as a dominator to a node Y if every path from the start node to Y goes through X. Similarly, a node X is defined as a post-dominator to a node Y if all paths to the exit node of the graph starting at Y go through X. The super-node method will reduce the proposed verification scheme overhead. Verifying at least one instruction from each super-node block will be sufficient to verify the entire slice. The following algorithm summarizes the steps needed for selecting the desired static slices.

Algorithm 1 Static Slices Selection

```
1: Generate Control Flow Graph CFG for $P$.
2: for each function $F$ in CFG
3:     Find all super-nodes in $F$ and add them to $N$.
4: end for
5: Randomly select a subset of super-nodes from $N$ and add
   them to $M$.
6: for each super-node in $M$
7:     Generate a backward static slice $SS$ that goes through
       the selected node and highlight these nodes.
8: end for
```

Probabilistic Verification Algorithms:

In this section, we propose use of a Bayesian inference on a binomial proportion method to verify the outsourced computation statistically. Bayesian inference is a statistical technique to update our subjective beliefs as new evidence or data becomes available. Our objective here is to characterize the probability density function for the outsourced computation correctness given that a set of slices were run correctly. In particular, we are interested in estimating confidence in verifying the correctness of the calculation results returned by an untrusted cloud server. Bayesian computation of probability distributions starts with a prior belief about a model parameter, then updates this distribution based on observed data to produce new posterior beliefs.

Before discussing the Bayesian procedure, we will state some underlying assumptions about our approach.

- We are going to assume that in verification of execution, the slice can only have two outcomes, pass or fail.
- Each slice generation is independent of the others, *i.e.* we use a random generator to identify the slicing criteria used in slice generation.
- Since we have no prior belief about the correctness of the results, we chose a uniform density of the beta distribution Beta (α, β) Gupta and Nadarajah (2004) to quantify our prior beliefs.

As stated above, we are interested in estimating the probability, given that a set of random slices passed the verification process, that the results returned by a server are correct. The mathematical definition of the Bayesian method is as follows: this method relies on Bayes

$$p(H|D) = \frac{p(D|H) \times p(H)}{p(D)} \tag{3.2}$$

Where H denotes our hypothesis, the correctness of the returned results, and D represents the evidence or the data, the number of slices that passed the verification.
$P(H|D)$ is the posterior probability distribution. This is the probability distribution of our belief of hypothesis H after evidence D has been taken into account.

$P(H)$ is the prior probability distribution, the strength in our belief of hypothesis H before any evidence D is observed.

$P(D|H)$ is the likelihood function, representing the probability of data D as generated by a set of models with parameter H. The probability density functions of the likelihood function can be chosen depending on the characteristics of the system, and in our approach, we propose to use the binomial distribution Annett (1967) since we assume that the verification of slice execution can only have two outcomes. The definition of the Bernoulli likelihood function is shown below:

$$p(D|H) = \binom{n}{y} H^{y}(1-H)^{(n-y)} \tag{3.3}$$

P(D) is the evidence probability distribution, the total probability of the data.

For simplicity, we will assume that P(D) =1, implying that any failure in one slice will cause rejection of the returned results. After applying the Bernoulli Function and the beta distribution Beta (α, β) in the Bayesian equation, we obtain the following equation:

$$p(H|\alpha, \beta) = \frac{H^{(\alpha-1)} \times (1-H)^{(\beta-1)}}{Beta(\alpha, \beta)} \tag{3.4}$$

Where α and β in terms of the Mean μ and the standard deviation σ is given by:

$$\alpha = (\frac{(1-\mu)}{\sigma^2} - \frac{1}{\mu}) \times \mu^2, \beta = \alpha \times (\frac{1}{\mu} - 1) \tag{3.5}$$

3.6 Evaluation of the Reversible SW-PUF

We evaluate of 32-bit Reversible SW-PUF in HSPICE k-2015.06 using predictive technology model HiSIM241 [13] for all NMOS/PMOS transistors with a V_{DD} of 3 V. The HiSIM241 model supports Design for Manufacturability (DFM). Matlab was used to generate random values for the DFM parameters for each transistor which creates 16 different patterns of PUF instances. We studied three metrics: uniqueness, randomness, and reliability.

3.6.0.1 Uniqueness

Uniqueness measures the capability to distinguish between different devices. Hamming Distances (HD) between PUF responses are used to measure uniqueness. An ideal HD between any two PUF responses is 50% (16-bit).

To evaluate the uniqueness of the Reversible SW-PUF on the same ALU under different data inputs (Intra-chip), we measured the average HD distribution between a pair of output data on the same device PUF instance with different set of input data.

The uniqueness of the forward signature for the Reversible SW-PUF has been measured the same way as the regular SW-PUF Hamadeh and Tyagi (2019). Figure 3.6 (A) shows that the maximum Hamming distance was 12 (38% of PUF response bits); the minimum was 3 (9% of PUF response bits); and the average was 6.6 (21% of PUF response bits). For the reverse computation, both ALUs inputs were measured on ten different PUF instances with identical output (response, which constitutes the input for a reversible PUF in reverse mode). The Hamming distance between each pair of different ALUs was calculated. Figure 3.6 (B) shows that the maximum Hamming distance was 14 (44% of PUF response bits); the minimum was 4 (13% of PUF response bits); and the average was 9.5 (30% of PUF response bits).

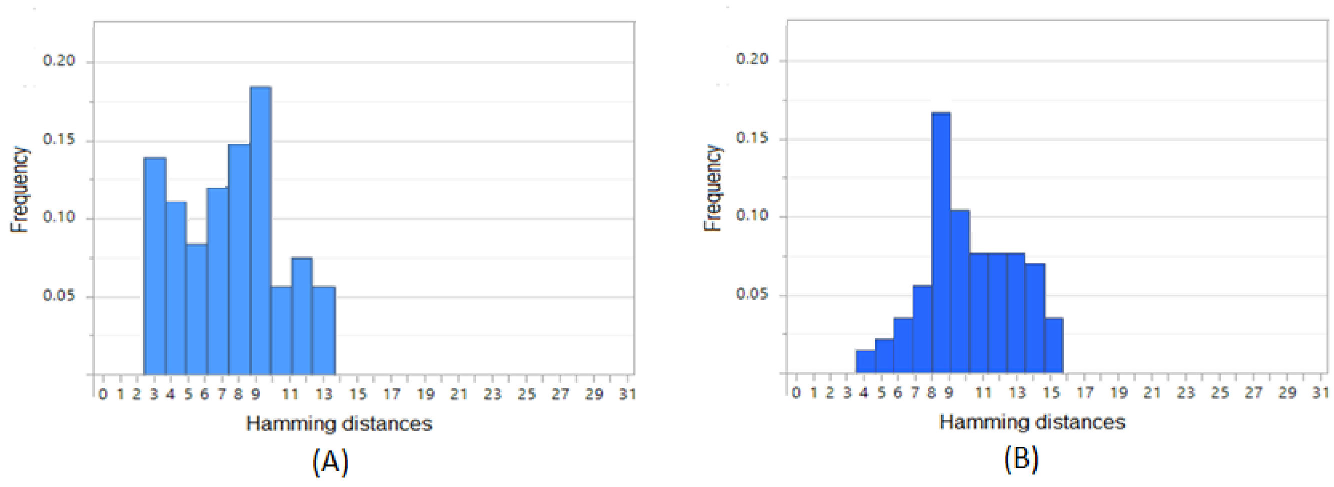


Figure 3.6 Hamming distance distribution of reversible SW-PUF: (A) Forward mode; (B) Reverse mode

3.6.0.2 Randomness

Randomness evaluates a PUF signature by analyzing the distribution of 0's and 1's. The standard statistical test suite of the National Institute of Standard and Technology (NIST) Bassham et al. (2010) was used to evaluate the responses of the reversible SW-PUF. We have applied the NIST tests to 512-bit stream that was produced from the 16 PUF instances. Table 3.1 shows the statistical tests results that were observed. Only two category (rank and linear complexity), out of fifteen statistical tests applied was failed.

Table 3.1 Summary results of statistical tests on reversible SW-PUF responses.

0	#Statistical Test	Avg. Success
1	Frequency	96.875%
2	Block Frequency	96.875%
3	Cumulative Sums	96.875%
4	Run	96.875%
5	Longest Run	96.875%
6	Rank	0%
7	Discrete Fourier Transform	100%
8	NonOverlappingTemplate	92.187%
9	OverlappingTemplate	100%
10	Universal	100%
11	RandomExcursionsVariant	100%
12	Approximate Entropy	100%
13	Serial	100%
14	LinearComplexity	0.8437%
The minimum pass rate for each statistical test is approximately 90.6%		

3.6.0.3 Reliability

Reliability measures robustness of a PUF in the presence of environmental variations. Temperature variations are the main factor that affect the stability of a PUF response. Therefore, reliability can be determined by varying the operating temperature and comparing the responses with a reference response obtained under nominal temperature conditions for the same input. We used the following equation for calculating the reliability of the reversible SW-PUF responses:

$$Reliability = \left(1 - \frac{1}{K}\sum_{i=0}^{n-1}\frac{HD(R_i, R_{i,j})}{n}\right) \times 100\% \tag{3.6}$$

where R_i is the reference response at room temperature 25°C, $R_{i,j}$ is a response observed under varying temperature value for the same challenge, n is the number of bits of reversible SW-PUF, and K is the number of samples. Figure 3.7 shows the reliability results for the responses of the

Reversible SW-PUF for both the forward and backward computations. The reversible PUF is very stable under the temperature variation from -10°C to 65°C.

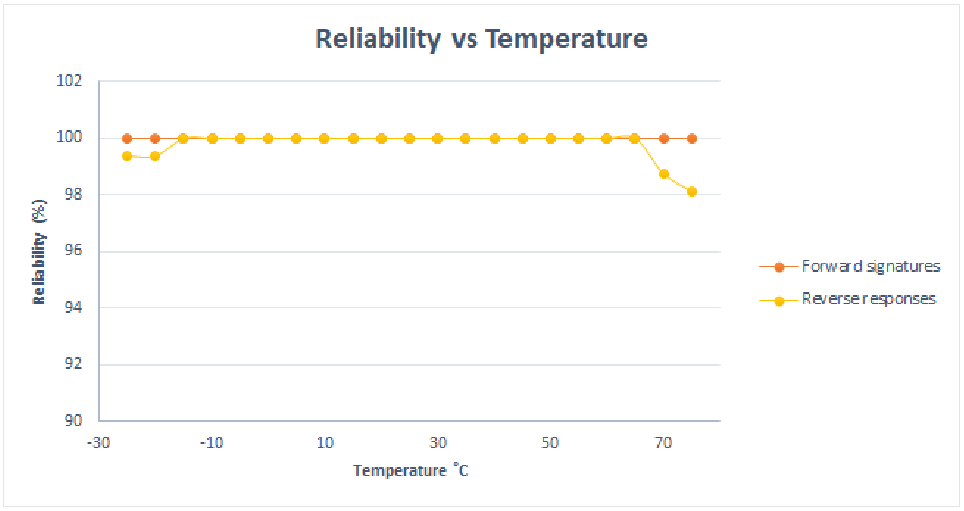


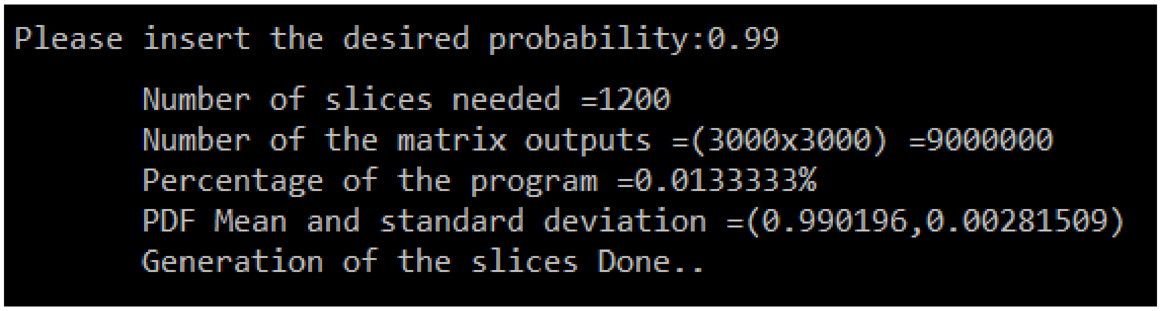
Figure 3.7 The reliability of reversible SW-PUF against temperature variations

3.7 Case Study: Verification of Matrix Multiplication

In this section, we evaluate the proposed method thorough a matrix multiplication experiment, a widely-used example in Verifiable Computation Systems Hui et al. (2018) , Zhang et al. (2017). We considered the following scenario: a client C needs to multiply two large scale matrices $A(n \times n)$and $B(n \times n)$ using a cloud service S. However, since the client C does not completely trust the cloud S to return the correct results for multiplication, the client C could verify the results in many ways. A naive algorithm could replicate the multiplication using another cloud service and compare the results, but this method is expensive, e.g., multiplying $n \times n$ matrices execute $O(n^3)$ time using the standard method. A faster check could use Freivalds' algorithm Freivalds (1979), a probabilistic randomized algorithm that verifies matrix multiplication in $O(kn^2)$ with a probability of failure less than 2^{-k}. Our approach improves Freivalds' algorithm by reducing the running time of the verification process by a factor of $O(n)$. Finally, we compare the execution time of our approach with the Verifiable Computation method proposed in Zhang et al. (2017).

3.7.0.1 Experimental Setup

We implemented a C++ tool to generate the random slices and perform the verification, and a LLVM compiler framework Lattner and Adve (2004) to compile the matrix multiplication program into LLVM Immediate Representation (IR). We used the Symbiotic 3 tool Chalupa et al. (2016) to obtain the backward static slice for the program. Symbiotic 3 linked with C++ code to generate the random slices in which the slicing criterion was one element of the output matrix. The number of slices was chosen based on the Bayesian Inference model. For simplicity, we assumed that client C challenges must completely match the server signatures, and any failure will result in rejection of the verification. Finally, a Pin tool Luk et al. (2005) was used to generate the desired instruction traces, while HSPICE was used to represent the Reversible SW-PUF to generate the signatures. Figure 3.8 shows an example of the proposed tool in which the Client C must choose the desired probability, after which the tool will generate the appropriate number of slices.


```
Please insert the desired probability:0.99

       Number of slices needed =1200
       Number of the matrix outputs =(3000x3000) =9000000
       Percentage of the program =0.0133333%
       PDF Mean and standard deviation =(0.990196,0.00281509)
       Generation of the slices Done..
```

Figure 3.8 An example of generating the desired slices

3.7.0.2 Performance evaluation

We performed the experiments for evaluating our scheme and present the computation time cost for each of its elements. The resultant time cost was obtained by averaging the outcomes of testing 10 different randomly generated inputs of the matrix multiplication code for matrix sizes ranging from 1000 to 7000.

As shown in Figure 3.9, the beta distribution can be used to model the prior and posterior of our beliefs. In Figure 3.9, we present three values of posterior beliefs; all of them start with the same prior (0.5, 0.1) this range is used to represent the absence of the prior information. The first posterior distribution is assessed after observing and verifying 100 slices; the probability of the total computation correctness was about 0.9. While observing and verifying 900 slices will increase the probability of the correctness to about 0.98.

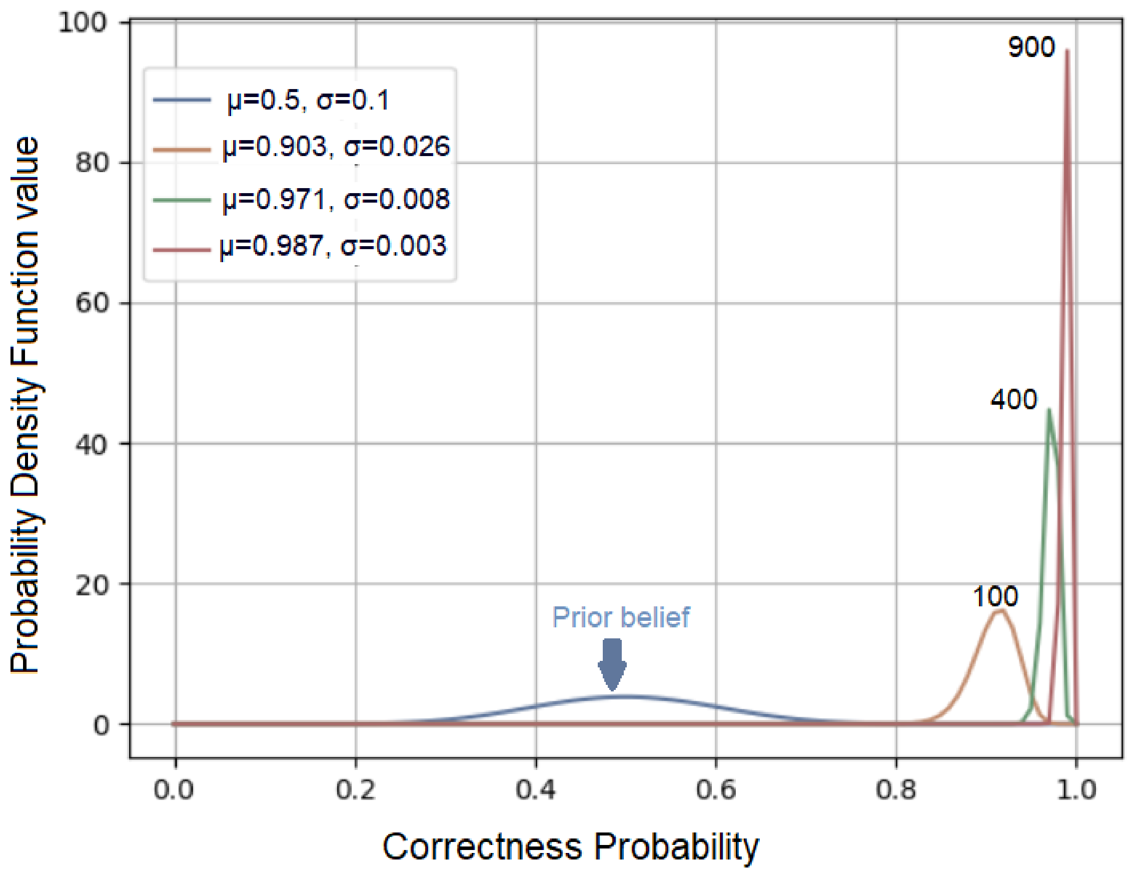


Figure 3.9 The prior and posterior belief distributions for various experiment

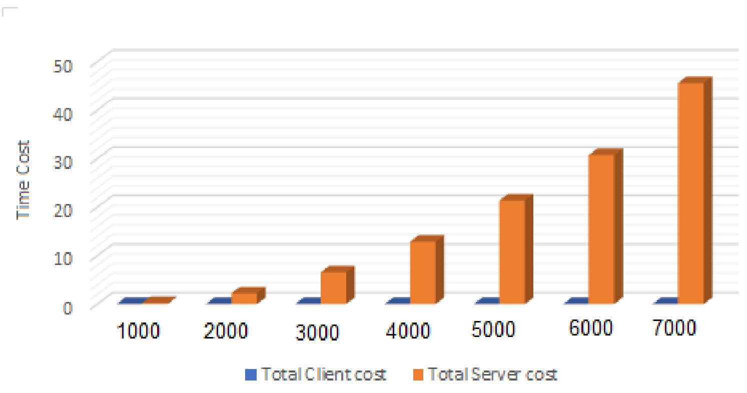


Figure 3.10 Computational time cost comparison between the Server and the Client sides

Table 3.2 shows a computational cost comparison between the server S (i.e. Matrix Multiplication, Reverse computations) and the client C (i.e. Slices Generation "the number of slices was picked to produce a probability of more than 0.97", Signatures Verification) sides. As we can see from Figure 3.10, the total computation time on the client C side was negligible compared to the total computation time on the server S side. For example, when $n = 1000$, the total time executed on the server S side was about $0.0209s$, while the total time needed on the client C side was only about $0.01s$. When $n = 7000$, the total time executed on the server S side was about $45.573s$, while the total time needed on the client C side was only about $0.021s$.

Table 3.2 Computation cost of proposed scheme for different problem size.

Dimension	Verification at Client Side		Computations at Server Side	
	Slices Generation	Signatures Verification	Matrix Multiplication	Reverse computations
n= 1000	10.025 ms	0.570 ms	0.201 s	0.008 s
n= 2000	11.504 ms	0.684 ms	2.129 s	0.078 s
n= 3000	12.753 ms	0.746 ms	6.372 s	0.183 s
n= 4000	15.025 ms	0.866 ms	12.479 s	0.366 s
n= 5000	16.875 ms	0.925 ms	20.692 s	0.675 s
n= 6000	17.752 ms	0.990 ms	29.668 s	1.065 s
n= 7000	20.057 ms	1.136 ms	44.050 s	1.523 s

We evaluate the advantage of our scheme by comparing our experiment with the PVCBMM scheme proposed in Zhang et al. (2017). Both of the experiments are performed on the same computer properties. However, we used the Strassen's algorithm Huss-Lederman et al. (1996) to reduce the time required to multiply matrices. We studied seven dimensions size ranging from 1000 to 7000. As shown in Table 3.3, the experimental results reveal that our scheme is more efficient than the PVCBMM scheme.

Table 3.3 Computation and Verification cost between two schemes.

Dimension	The proposed scheme		PVCBMM scheme Zhang et al. (2017)	
	Computations cost	Verification cost	Computations cost	Verification cost
n= 1000	0.201 s	0.018 s	1.75 s	6.94 s
n= 2000	2.12 s	0.090 s	4.36 s	14.86 s
n= 3000	6.37 s	0.19 s	8.35 s	32.26 s
n= 4000	12.47 s	0.38 s	24.62 s	61.37 s
n= 5000	20.69 s	0.69 s	36.31 s	85.03 s
n= 6000	29.66 s	1.08 s	65.16 s	178.54 s
n= 7000	44.05 s	1.54 s	105.28 s	193.86 s

3.8 Adversary Model

There are many possible threat behaviors that can cause a server cloud node to return incorrect results, and in this paper, we will consider the following:

Dishonest Server: In this model the server may alter the inputs to reduce the number of computations.

Hardware or Software Failures: An error may occur during the computation and the server will not redo the computation but instead present fake results.

Malicious attacks: An unauthorized attacker may be able to access the cloud node and inject malicious code that will affect result quality.

All these models can be discovered using the proposed scheme, so the client can detect a mismatch in the returned signatures with high probability.

3.9 Conclusions

We present reversible SW-PUF, a novel PUF design for computing partial inputs given a set of outputs. We implemented the reversible SW-PUF in HSPICE and established its desirable properties (uniqueness, randomness, and reliability). We then provided an efficient interactive verifiable computation scheme based on the proposed PUF and based on the Bayesian method. Our approach links outsourced computation with server cloud node hardware to provide proof of correctness of the results with high probability. We evaluated our verifiable computation scheme and demonstrated that it yields faster verification than previous approaches.

3.10 References

Annett, M. (1967). The binomial distribution of right, mixed and left handedness. *The Quarterly journal of experimental psychology*, 19(4):327–333.

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., Heckert, N. A., Dray, J. F., and Vo, S. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Gaithersburg, MD, United States.

Bennett, C. H. (1973). Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532.

Binkley, D., Gold, N., and Harman, M. (2007). An empirical study of static program slice size. *ACM Trans. Softw. Eng. Methodol.*, 16(2).

Chalupa, M., Jonáš, M., Slaby, J., Strejček, J., and Vitovská, M. (2016). Symbiotic 3: New slicer and error-witness generation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 946–949. Springer.

Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R., and Stüble, C. (2006). A protocol for property-based attestation. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, STC '06, pages 7–16, New York, NY, USA. ACM.

Choi, P. and Kim, D. K. (2012). Design of security enhanced tpm chip against invasive physical attacks. In *2012 IEEE International Symposium on Circuits and Systems*, pages 1787–1790. IEEE.

Dempster, A. P. (1968). A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232.

Dey, B., Khalil, K., Kumar, A., and Bayoumi, M. (2019). A novel design gate based low-cost configurable ro puf using reversible logic. In *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 211–214. IEEE.

Fredkin, E. and Toffoli, T. (1982). Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253.

Freivalds, R. (1979). Fast probabilistic algorithms. In *International Symposium on Mathematical Foundations of Computer Science*, pages 57–69. Springer.

Gupta, A. K. and Nadarajah, S. (2004). *Handbook of beta distribution and its applications*. CRC press.

Hamadeh, H. and Tyagi, A. (2019). Physical unclonable functions (pufs) entangled trusted computing base. In *2019 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*. IEEE.

Hui, L., Dong, X., Shen, J., Cao, Z., Chen, H., and Liang, Y. (2018). Vepp: A verifiable, highly efficient and privacy-preserving protocol for outsourcing large matrix multiplication. In *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–8. IEEE.

Huss-Lederman, S., Jacobson, E. M., Johnson, J. R., Tsao, A., and Turnbull, T. (1996). Implementation of strassen's algorithm for matrix multiplication. In *Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 32–32. IEEE.

Lattner, C. and Adve, V. (2004). Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society.

Lebedev, I., Hogan, K., and Devadas, S. (2018). Secure boot and remote attestation in the sanctum processor. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 46–60. IEEE.

Li, D., Guo, H., and Xu, J. (2016). Enhancing tpm security by integrating sram pufs technology. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 82–93. ACM.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200. ACM.

Müller, K.-U., Ulrich, R., Stanitzki, A., and Kokozinski, R. (2018). Enabling secure boot functionality by using physical unclonable functions. In *2018 14th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, pages 81–84. IEEE.

Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, Washington, DC, USA. IEEE Computer Society.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2016). Pinocchio: Nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112.

Sailer, R., Zhang, X., Jaeger, T., and van Doorn, L. (2004). Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 16–16, Berkeley, CA, USA. USENIX Association.

Thaler, J., Roberts, M., Mitzenmacher, M., and Pfister, H. (2012). Verifiable computation with massively parallel interactive proofs. In *HotCloud*. USENIX Association.

Toffoli, T. (1980). Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644, London, UK, UK. Springer-Verlag.

Vu, V., Setty, S. T. V., Blumberg, A. J., and Walfish, M. (2013). A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 223–237. IEEE Computer Society.

Weiser, M. (1981). Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE '81, pages 439–449, Piscataway, NJ, USA. IEEE Press.

Zhang, M., Gu, Z., Li, H., and Zheng, N. (2018). Wcet-aware control flow checking with super-nodes for resource-constrained embedded systems. *IEEE Access*, 6:42394–42406.

Zhang, X., Jiang, T., Li, K.-C., Castiglione, A., and Chen, X. (2017). New publicly verifiable computation for batch matrix multiplication. *Information Sciences*.

CHAPTER 4. PRIVACY PRESERVING DATA PROVENANCE MODEL BASED ON PUF FOR SECURE INTERNET OF THINGS

Modified from a paper accepted by *IEEE-iSES 2019 conference*

Hala Hamadeh and Akhilesh Tyagi

4.1 Abstract

Data provenance to maintain data integrity and authenticity is a significant challenge in the Internet of Things (IoT) environments. Additionally, if the provenance metadata itself can be communicated in a privacy preserving manner, it expands the usage of IoT systems to human societal domains where privacy is of paramount importance. In this paper, we present a scheme to combine data provenance and privacy-preserving solutions. Our scheme merges Physical Unclonable Function (PUF) technology with non-interactive zero-knowledge proof to provide trustworthy and dependable IoT systems. In this context, the IoT device can anonymously send data to the corresponding server associated with the proof of ownership. First, we propose a privacy-preserving data provenance protocol. This protocol was synthesized with Altera Quartus. It was implemented on an Altera Cyclone IV FPGA to demonstrate its practicality and feasibility. Most of the protocol steps take time of the order of 40μ sec establishing its practicality.

4.2 Introduction

The Internet of Things (IoT) technology deployment has been growing exponentially within the last decade Conti et al. (2018); Srinivasan et al. (2019); Lu and Da Xu (2018). IoT's are everywhere from a smart and connected home, to hospitals, to military & agriculture Firouzi et al. (2018); Ray (2018); Abdel-Basset et al. (2019). This is still the proverbial tip of an iceberg. The ceiling for IoT deployment still has much further to go. This growth brings along several challenges, specially in the area of cyber-security.

Provenance and privacy preservation are considered two important factors within IoT cber-security domain due to the fact that the data is transmitted over communication channels. More specifically, in an IoT system, data provenance refers to the metadata that describes the ownership, creation process, and modification of data. Providing secure data provenance aims to establish the trust in the data collected among the IoT devices Aman et al. (2017). Moreover, since IoT networks are ideally open systems to allow plug-in functionality extension, the data provenance should be communicated in a way so that the privacy of the provenance provider is not violated by leaking un-necessary information. This is what a privacy preserving data provenance model seeks to establish.

Physical Unclonable Functions (PUF) are good candidates for providing a unique device-specific identity. Such unique silicon biometric identities can be a good source of data provenance. Software PUF (SW-PUF) Hamadeh and Tyagi (2019) composes the silicon fabrication process variation with the software input defined execution paths to generate reproducible randomness that is both device and software dependent to serve as a hardware-software fingerprint. This functionality allows the SW-PUF to provide unique metadata to certify if a specific IoT device executed a specific data creation or modification program Hamadeh and Tyagi (2019).

Privacy-preservation deals with protection of the IoT devices' identities. One mechanism to keep the identity private is based on non-interactive Zero-Knowledge Proof Rackoff and Simon (1991). Furthermore, integration of the data provenance with the privacy-preserving protocols is expected to provide a significant benefit in many IoT scenarios. For instance, consider health-care monitoring in an elder care center. Many vital signs for the elderly patients, such as heart rate, respiration flow rates, and temperature, could be tracked. In this case, trust in the received data could derive from trust in the identity of properly registered IoT sensors. However, if the data transmission protocol reveals the identity of the specific IoT sensors, it automatically reveals the patient identity. Any exposure of such sensitive data by unauthorized parties is considered as a violation of federal health insurance portability and accountability act rules. Moreover, any tampering or modification of this data can lead to fatal outcomes Adhikary et al. (2019). This makes trust in the received data from the IoT sensors very critical. We believe a privacy preserving data provenance model provides backbone for such a trust model.

In this work, we propose a novel privacy preserving data provenance model based on Physical Unclonable Functions and Non-Interactive Zero-Knowledge Proof systems. This framework guarantees that the received data from an IoT device is collected from a registered authorized device; that it can be verified that the said authorized device ran a specific authorized data creation or modification program; and that the preceding two properties can be established without revealing the device identity. Specifically, the proposed solution contributes to achieving the following security goals:

- Source Identity Authenticity: guarantees that the data originated from the specific IoT device that sent it.
- Privacy-Preserving Identity: ensures that the real identity of the owner of the data is not unveiled.
- Data Integrity: confirms that the data transmitted is not tampered with.
- Device Trust: ensures that the device is not exploited by a malicious code.

4.3 Related Work

Recently, several data provenance protocols have been investigated in literature Kamal et al. (2018); Elkhodr et al. (2018); Jaigirdar et al. (2019). However, only a few of these studies focus both on the data provenance and privacy-preserving protocols for an IoT system.

For instance, in Sanchez et al. (2018), the authors extended the IBM Idemix protocol with a non-interactive zero-knowledge proof to sign the metadata transmitted by an IoT device in a privacy-preserving way. However, this solution is limited to only some types of systems. The work in Alharbi and Lin (2012), investigates the privacy-preserving data provenance. However, their design relies on the trust of the server itself on an ecosystem. Few other techniques establish secure data provenance based on PUFs. The authors in Javaid et al. (2018) proposed to use PUF with blockchain network in order to add a unique hardware fingerprint to each IoT device. Their proposed solution depends on storing many Challenge Response Pairs (CRPs) in the server memory exposing them to storage attack concerns. In Aman et al. (2017), Aman et. al have used the Received Signal Strength Indicator (RSSI) along with PUF and symmetric encryption to provide the privacy of the transmitted metadata between the IoT device and the server. Similar to the work in Javaid et al. (2018), this scheme also relies on storing many CRPs in the server; in addition it also relies on the RSSI values which is not practical in some IoT environments. On the contrary, our solution attempts to overcome all these issues.

4.4 Privacy Preserving data provenance protocol

In this section, we describe our scheme for secure data provenance in an IoT system. The proposed protocol encompasses four stages. The first stage (called the Setup and Enrollment stage) is to generate public parameters required by the following stages in the protocol. These parameters are associated with an IoT device profile indexed by a virtual ID assigned to an IoT device by the server. The second stage, called authentication stage, is to prove the identity of an IoT device to the server ensuring its privacy. The third stage, called Key Exchange stage, is to exchange a symmetric key between a server and an IoT device. The last stage, called Data Transmission and Verification stage, is to start trusted communication between an IoT device and the server which confirms the source of the transmitted data. Our design is based on a software PUF and non-interactive zero-knowledge proofs. The SW-PUF Hamadeh and Tyagi (2019) is used to provide a proof of identity and root of trust for an IoT device. This type of PUF ensures that the data generated and processed by an IoT device is measured as it is computed inside the IoT device itself. This is beneficial in proving the provenance of the data. Non-interactive zero-knowledge proofs were used to ensure secure privacy-preserving communication between an IoT device and a server in the authentication process. The elliptic curve cryptography over Binary Fields GF(2^m)

was chosen in order to reduce the computational requirements for the IoT devices while maintaining the security level of other mathematical frameworks.

A SW-PUF exploits the fact that in an ALU (carry-ripple adder like design), the delays to settle the output (s_i) or carry (c_i) bits are data-dependent. For some input combinations for $A + B$ the carry-chain takes only time proportional to one bit-slice, whereas for some others it takes time proportional to n bit slices (for n bits input data). Average case delay is proportional to $O(\log n)$. Similarly, the delays are silicon foundry dependent along the lines of all PUFs due to process variations. In synchronous designs, we pick a clock period corresponding to worst case delay of $O(n)$. However, if we pick an output sampling period closer to $\log N$, we see both input data and silicon dependent randomness. This serves as a software PUF that verifies both the device identity through silicon randomness and software identity through execution paths leading to a specific sequence of data inputs. If such an adder were to be build with reversible logic, such as transmission gate logic, with conservative logic such as Fredkin-Toffoli gates, it can also certify its identity to a third party through an interactive challenge-response verification. Its responses Ri to various input combinations Ci can be recorded as (Ci, Ri) CRPs. If the verifier has access to these (Ci, Ri) CRPs, it can randomly pick a challenge Ci and ask the SW-PUF to generate the Ci in the reverse mode. If the protocol ensures that the prover device is not able to use precached (Ci, Ri) pairs to respond, we can ensure that the SW-PUF exists at the device and behaves as anticipated.

4.4.0.1 Enrollment and Setup stage

This stage is performed only once when an IoT device is deployed in the field for the first time. An IoT device and a server prepare all parameters required to perform the authentication protocol in the future and agree on a virtual ID for communication with the IoT device. The following steps will be performed by an IoT device to generate public parameters required by the following stages: Note, none of the parameters generated in this stage contain sensitive information, thus, it is safe to transmit them over an open communication channel.

Device: • Select elliptic curve E over Binary Fields GF(2^m)

- Choose base point $G = (Gx, Gy)$. Not that in ECC, many parameters like G are points in 2-dimensional space. We will often refer to the x and y components of such points by notation Gx and Gy respectively.
- Compute public Key $A = x \cdot G$ where x is the SW-PUF signature that has been generated during the bootup of the device.
- Share the public parameters $\{G = (Gx, Gy), A = (Ax, Ay)\}$ with the server.

Server:

- Generate Virtual id (Vid) for the device to use in future communications.
- Store the public parameters associated with this Vid.

4.4.0.2 Authentication stage

The authentication of the identity of an IoT device could be performed based on SW-PUF Hamadeh and Tyagi (2019) and non-interactive zero-knowledge proofs like IBM Idemix Camenisch and Van Herreweghen (2002), Wallrabenstein (2016). Our protocol uses a unique SW-PUF signature (x) that can be generated during the bootup phase of the device to authenticate the execution environment of the device, or it can be generated every time a new data is produced or processed to authenticate the data creation or modification step at a specific IoT device. If a log of a sequence of data creation and modification events at a specific IoT device needs to be authenticated, then a chained hash of these raw SW-PUF signatures in the order of events' occurrence is needed. This is similar to the way a TPM maintains platform configuration registers (PCRs). Then, this signature hash needs to be saved in a protected memory in the IoT device that we will refer to as Metadata Tracking Register (MTR). MTR's role is similar to the TPM's PCR. It holds a chained hash of provenance metadata evolution through creation and modification steps. The MTR can only be updated through MTR extension API which takes the hash of current MTR value concatenated with the new SW-PUF signature val as the new MTR value $MTR \leftarrow h(MTR(v)||val)$, same as the PCR extension. Figure 4.1 shows the proposed protocol for this stage.

The authentication protocol is based on non-interactive zero-knowledge proofs Rackoff and Simon (1991). These protocols can prove the knowledge or possession of a value to a verifier without requiring multiple interactive steps (as in traditional zero-knowledge proofs). This verification leaks zero information about the value known to the prover. Let the value known to the prover be x. The prover generates two derived values from x, a t-value given by $t = f_t(x, K)$ - a function of the secret value x and several public parameters such as a key K, and potentially others such as nonces; a s-value given by $s = f_s(x, K)$ - a function of the secret value x and several public parameters such as a key K, and potentially others such as nonces. The functions f_s and f_t allow the verifier to check on some mathematical properties of s and t combined which is not likely to hold unless the prover knows x. But s and t together do not reveal x. The IoT device (Prover) performs the following steps every time the device boots up to start a trusted communication with the server (Verifier):

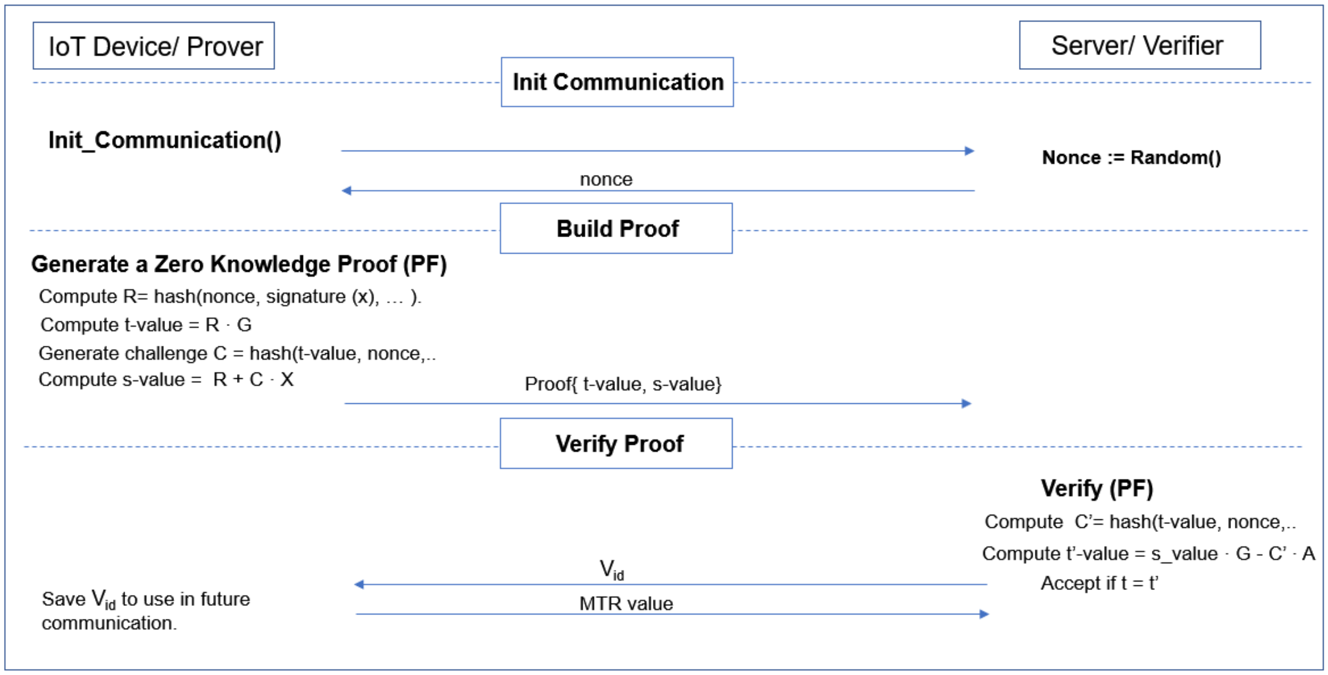

IoT Device/ Prover
Server/ Verifier
Init Communication
Init_Communication()
Nonce := Random()
nonce
Build Proof
Generate a Zero Knowledge Proof (PF)
Compute R= hash(nonce, signature (x), …).
Compute t-value = R · G
Generate challenge C = hash(t-value, nonce,..
Compute s-value = R + C · X
Proof{ t-value, s-value}
Verify Proof
Verify (PF)
Compute C'= hash(t-value, nonce,..
Compute t'-value = s_value · G - C' · A
Accept if t = t'
V_id
MTR value
Save V_id to use in future communication.

Figure 4.1 Authentication protocol

Prover: • Ask the Server (Verifier) to initiate the communication.

Verifier: • Send a random nonce to the prover to ensure that old communications cannot be reused.

Prover: • Compute $v = hash(\ nonce || \ x || \ all\ public\ parameters || \dots)$.

- Compute *t-value* where $t = v \cdot G$. Note that G is a pulic parameter generated during enrollment.
- Generate challenge $c = hash(t\text{-}value\ ||\ \ all\ public\ parameters\ || \dots)$.
- Compute *s-value* where $s = v + c \cdot x$.
- Send *t-value*, *s-value* to the verifier.

Verifier: • Compute $c' = hash\ (t\text{-}value || \ all\ public\ parameters || \dots)$

- Compute $t'\text{-}value = s \cdot G - c' \cdot A$. Note that A is a public key generated and published during enrollment.
- Verify that $t\text{-}value = t'\text{-}value$ holds.
- Reject in case of a mismatch.

Once the server (verifier) authenticates the IoT device (prover) s/he sends Virtual id (Vid) to the device for future communications. Note that Vid was generated during enrollment, but not shared with the IoT device until after authentication. The IoT device could update the metadata tracking register (MTR) value as follows: $MTR_{\text{new}} \leftarrow hash(MTR_{\text{old}} = SW - PUF\ at\ bootup)$. This version of MTR however reveals the raw SW-PUF bootup signature. We need to hide it with some other random parameters. We, however, cannot use non-reproducible parameters such as time or a random nonce. An MTR digest should be reproducible for the same sequence of bootup, data creation, and data modification events for the verifiability, just as PCR digests are. One solution to this is to create a special program module called *nonce-parameter-generator (seed)*. When this program executes, SW-PUF generates its signature, which can be used as a nonce like hiding parameter. The *seed* could be the bootup SW-PUF signature in MTR_{old}, which should be reproducible for the future device boot ups. This is the version we propose to use:

$MTR_{\text{new}} \leftarrow hash(MTR_{\text{old}}||nonce - parameter - generator(MTR_{\text{old}}))$. The prover/device sends this MTR_{new} to the server/verifier to be used in the future steps.

4.4.0.3 Key Exchange stage

To ensure secure communication between the IoT device and the server, we use symmetric AES encryption algorithm Srinivas and Akramuddin (2016) for better efficiency in place of an asymmetric encryption system. In this section, we explain the key agreement process between the device and the server. We use a hash function with a standard key exchange based on elliptic curve Diffie-Hellman protocol Ahirwal and Ahke (2013) to generate and exchange the AES key between the two parties. The following explanation outlines the main steps:

Device:
- Choose private key k_{d} where $k_{\text{d}} < 2^m$.
- Compute public Key $Q_{\text{d}} = k_{\text{d}} \cdot G$. Recall that m and G are public parameters.
- Share the public parameters $(Q_{\text{d}}x, Q_{\text{d}}y)$ with the server.

Server:
- Choose private key k_{s} where $k_{\text{s}} < 2^m$.
- Compute public Key $Q_{\text{s}} = k_{\text{s}} \cdot G$
- Share the public parameters $(Q_{\text{s}}x, Q_{\text{s}}y)$ with the IoT device.

Device:
- Compute shared Key $K_{\text{ds}} = k_{\text{d}} \cdot Q_{\text{s}}$
- Compute AES key $K_{\text{AES}} = hash(K_{\text{ds}})$

Server:
- Compute shared Key $K_{\text{ds}} = k_{\text{s}} \cdot Q_{\text{d}}$
- Compute AES key $K_{\text{AES}} = hash(K_{\text{ds}})$

4.4.0.4 Data Transmission and Verification stage

In this section, we propose a transmission and verification protocol for securing the data transmission and providing verification of the provenance in IoT environments. As we stated earlier, the

AES algorithm has been chosen to encrypt/decrypt all the transmitted messages. The proposed protocol consists of three phases: Init Communication: where the IoT device initiates the communication by sending the Vid and the stored MTR value. Generating Data: where the IoT device generates both the data and the metadata and sends them the server. Verify Data Provenance: where the server verifies the data by confirming the source of the data. The proposed protocol is shown in Figure 4.2. The following steps describe each phase. This protocol needs to be repeated for every data transmitted from an IoT device to a server:

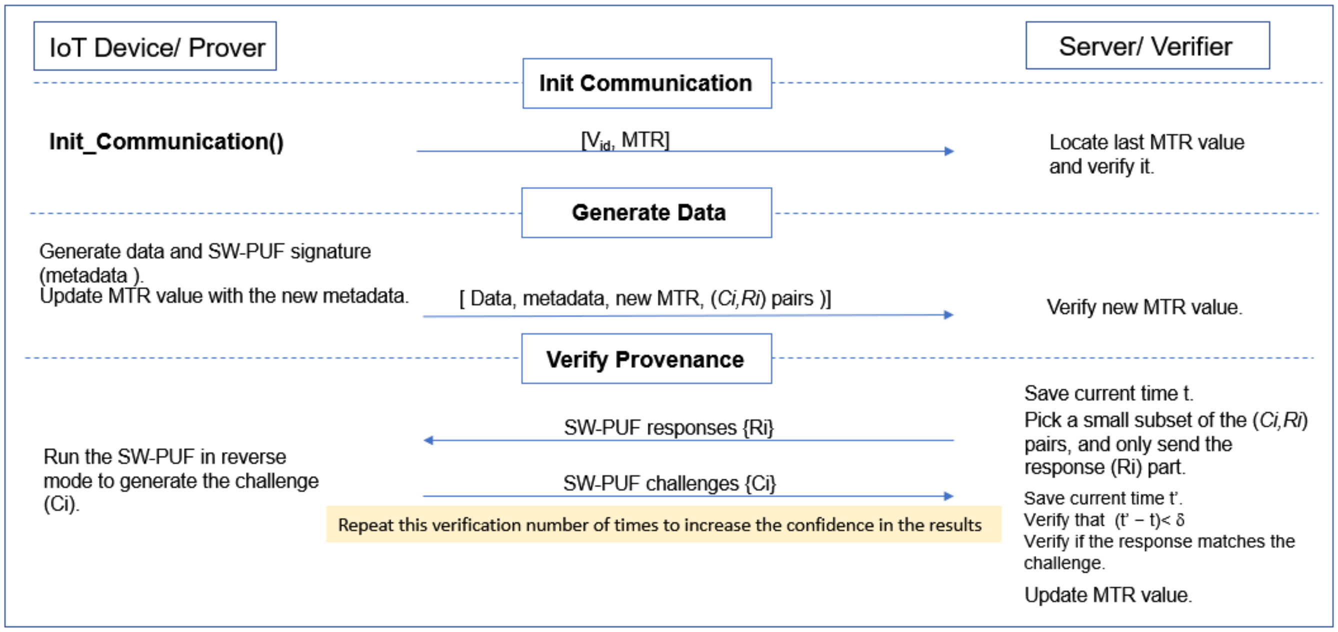

IoT Device/ Prover
Server/ Verifier
Init Communication
Init_Communication()
[V_id, MTR]
Locate last MTR value and verify it.
Generate Data
Generate data and SW-PUF signature (metadata).
Update MTR value with the new metadata.
[Data, metadata, new MTR, (Ci,Ri) pairs)]
Verify new MTR value.
Verify Provenance
Save current time t.
Pick a small subset of the (Ci,Ri) pairs, and only send the response (Ri) part.
SW-PUF responses {Ri}
Run the SW-PUF in reverse mode to generate the challenge (Ci).
SW-PUF challenges {Ci}
Repeat this verification number of times to increase the confidence in the results
Save current time t'.
Verify that (t' − t)< δ
Verify if the response matches the challenge.
Update MTR value.

Figure 4.2 Data Transmission and Verification protocol

Device:
- Ask the Server (Verifier) to initiate the communication.
- Send Vid and current MTR value.

Server:
- Locate the $MTRvalue$ for the received Vid and verify it.
- Reject in case of mismatch.

Device: • Generate the data and SW-PUF signature (metadata) of a data creation/modification event generated by the SW-PUF.

- Update MTR value with the new metadata.
  $MTR_{\text{new}} \leftarrow hash(MTR_{\text{old}}||SW\ PUF\ signature)$.
- Send Data and metadata consisting of MTR_{new} and SW-PUF (Ci, Ri) pairs to the server.

Server: • Verify the received metadata MTR_{new} and $SW\ PUF\ signature$ against the old value MTR_{old} associated with Vid by checking the equality $MTR_{\text{new}} = hash(MTR_{\text{old}}||SW\ PUF\ signature)$.

- Reject in case of mismatch.
- Save current time t.
- Pick a small subset of the helper SW-PUF (Ci, Ri) pairs. Verify the existence and integrity of SW-PUF by sending the response Ri part of the selected subset asking the prover to generate the corresponding Ci part through reversible computation.

Device: • Run the SW-PUF in reverse mode to generate the challenge Ci corresponding to the received Ri.

- Send Ci to the server.

Server: • Save current time t'.

- Verify that $(t' - t) < \delta$. This check ensures that Ci is computed in a reverse mode. An untruthful device would have needed more than δ time to perform additional computations or to retrieve it from a secondary storage. Note that we also assume that relative to the IoT device small cache, the (Ci, Ri) sets are significantly larger - preventing a cached response to bypass reverse computation.
- Verify if the received response $Ri_{received}$ matches the response Ri from the (Ci, Ri) pairs received at the MTR verification stage.
- Reject in case of mismatch.

- Update MTR value.

4.5 Threat model

Following are some of the objectives of an attacker for the proposed protocol:

- Mimic an IoT device and transfer maliciously modified data to the server.

  This attack cannot be applicable in our scheme since the IoT device identity is based on PUF which makes it close to impossible to generate or clone a fake identity.

- Tamper or modify the data sent by a valid IoT device.

  This attack can be detected by the proposed verification protocol, where the MTR value will not match the saved MTR value at the server.

4.6 Implementation

Complete design of the proposed privacy-preserving data provenance protocol has been modeled in Verilog (HDL), simulated by ModelSim XE, synthesis with Altera Quartus and implemented on FPGA Altera Cyclone iv at speed of 50 MHz. The AES-128 encryption and decryption algorithm, SHA-256 hash function, ECC over Binary Fields GF(2^{233}) engine. All the other proposed protocols in this paper also map to the FPGA. The SW-PUF is the only component not mapped to the FPGA since reversible computation using transmission gate logic is not feasible in an FPGA fabric. The data and the metadata (SW-PUF signatures) generation was performed in software using HSPICE k-2015.06 and Pin tool (dynamic binary instrumentation tool) Reddi et al. (2004).

As we explain earlier, the public-key cryptography used in the enrollment and authentication stages was based on an Elliptic curve over the binary field. The ECC is suitable for resource-constrained system because it can offer the same security level as other asymmetric systems for a much smaller key size. This implementation used the recommended Curve B-233 presented in the NIST FIPS Locke and Gallagher (2009) to provide excellent security level. The 233 bits key size has performance comparable to RSA 2048 bits key size.

4.7 Results

Our implementation consumes around 40K Logic Elements (LEs) for the IoT device and around 37K LEs for the IoT server. Table 4.1 reports the LEs needed for each step in our protocol. It takes around 118μ sec to perform the enrollment, authentication, and key exchange protocols. About 120m sec is required to transfer and to verify the provenance of 1 megabyte of data and metadata. Table 4.2 records the average execution time for each step in our protocol. Note that all the verification steps take about the same time, but the transmission & verification time dominates. Note: Communication time between the device and the server is not included, and it depends on the desired communication technologies. For example, Wi-Fi can transmit data at throughput up to 1 Gbps Li et al. (2018).

Table 4.1 Performance Results (Total Logic Elements (LEs))

Protocol	IoT Server	IoT Device
Enrolment	-	109
Authentication	$1,693$	729
Key Exchange	$22,941$	$22,941$
Transmission and verification	$16,670$	$14,170$

Table 4.2 Performance Results (Execution Time)

Protocol	IoT Server	IoT Device
Enrolment	-	$35.4u\ sec$
Authentication	$40.1u\ sec$	$40.0u\ sec$
Key Exchange	$42.5u\ sec$	$42.5u\ sec$
Transmission and verification	$120.2m\ sec$	$71.6m\ sec$

4.8 Conclusions

In this paper, we present a privacy-preserving data provenance solution that merges Physical Unclonable Function (PUF) technology with non-interactive zero knowledge proof to provide trustworthy and dependable IoT systems. In our scheme, an IoT device can anonymously send data to an IoT server. The server enrolls an IoT device and verifies all the provenance metadata for data creation and modification. The proposed protocol has been designed and synthesized with Altera Quartus and implemented on FPGA Altera Cyclone iv. The implementation demonstrates the practicality and feasibility of our solution, while the simulation results achieve practical performance that can be deployed for Resource-Constrained IoT Devices.

A server mediated trust system could potentially become a bottleneck in IoT system scalability. In the future, we are studying distributed systems that provide data provenance within a private framework.

4.9 References

Abdel-Basset, M., Manogaran, G., Mohamed, M., and Rushdy, E. (2019). Internet of things in smart education environment: Supportive framework in the decision-making process. *Concurrency and Computation: Practice and Experience*, 31(10):e4515.

Adhikary, T., Jana, A. D., Chakrabarty, A., and Jana, S. K. (2019). The internet of things (iot) augmentation in healthcare: An application analytics. In *International Conference on Intelligent Computing and Communication Technologies*, pages 576–583. Springer.

Ahirwal, R. R. and Ahke, M. (2013). Elliptic curve diffie-hellman key exchange algorithm for securing hypertext information on wide area network. *International Journal of Computer Science and Information Technologies*, 4(2):363–368.

Alharbi, K. and Lin, X. (2012). Pdp: A privacy-preserving data provenance scheme. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 500–505. IEEE.

Aman, M. N., Chua, K. C., and Sikdar, B. (2017). Secure data provenance for the internet of things. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, pages 11–14. ACM.

Camenisch, J. and Van Herreweghen, E. (2002). Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30. ACM.

Conti, M., Dehghantanha, A., Franke, K., and Watson, S. (2018). Internet of things security and forensics: Challenges and opportunities.

Elkhodr, M., Alsinglawi, B., and Alshehri, M. (2018). Data provenance in the internet of things. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 727–731. IEEE.

Firouzi, F., Rahmani, A. M., Mankodiya, K., Badaroglu, M., Merrett, G. V., Wong, P., and Farahani, B. (2018). Internet-of-things and big data for smarter healthcare: from device to architecture, applications and analytics.

Hamadeh, H. and Tyagi, A. (2019). Physical unclonable functions (pufs) entangled trusted computing base. In *2019 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*. IEEE.

Jaigirdar, F. T., Rudolph, C., and Bain, C. (2019). Can i trust the data i see?: A physician's concern on medical data in iot health architectures. In *Proceedings of the Australasian Computer Science Week Multiconference*, page 27. ACM.

Javaid, U., Aman, M. N., and Sikdar, B. (2018). Blockpro: Blockchain based data provenance and integrity for secure iot environments. In *Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems*, pages 13–18. ACM.

Kamal, M. et al. (2018). Light-weight security and data provenance for multi-hop internet of things. *IEEE Access*, 6:34439–34448.

Li, S., Da Xu, L., and Zhao, S. (2018). 5g internet of things: A survey. *Journal of Industrial Information Integration*, 10:1–9.

Locke, G. and Gallagher, P. (2009). Fips pub 186-3: Digital signature standard (dss). *Federal Information Processing Standards Publication*, 3:186–3.

Lu, Y. and Da Xu, L. (2018). Internet of things (iot) cybersecurity research: a review of current research topics. *IEEE Internet of Things Journal*, 6(2):2103–2115.

Rackoff, C. and Simon, D. R. (1991). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 433–444. Springer.

Ray, P. P. (2018). A survey on internet of things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3):291–319.

Reddi, V. J., Settle, A., Connors, D. A., and Cohn, R. S. (2004). Pin: A binary instrumentation tool for computer architecture research and education. In *Proceedings of the 2004 Workshop on Computer Architecture Education: Held in Conjunction with the 31st International Symposium on Computer Architecture*, WCAE '04, New York, NY, USA. ACM.

Sanchez, J. L. C., Bernabe, J. B., and Skarmeta, A. F. (2018). Towards privacy preserving data provenance for the internet of things. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 41–46. IEEE.

Srinivas, N. S. and Akramuddin, M. (2016). Fpga based hardware implementation of aes rijndael algorithm for encryption and decryption. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 1769–1776. IEEE.

Srinivasan, C., Rajesh, B., Saikalyan, P., Premsagar, K., and Yadav, E. S. (2019). A review on the different types of internet of things (iot). *Journal of Advanced Research in Dynamical and Control Systems*, 11(1):154–158.

Wallrabenstein, J. R. (2016). Practical and secure iot device authentication using physical unclonable functions. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 99–106. IEEE.

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis we developed and analyzed tools and protocols that provide hardware security. In Chapter 2 we presented new PUF architectures and evaluated some of their features that included uniqueness, reliability, and randomness. We also explored some of the applications of these architectures. We first presented a software PUF suitable for establishing a Software Protection and Trusted Computing Base. To validate PUF behavior, we implemented our design on an FPGA platform (Altera DE2 Development and Education Board). A simulation analysis in HSPICEk-2015.06 using 45 nm technology was also performed. The responses measured from the software PUF have a sufficiently good uniqueness of about 30% (50%) and more than 96% reliability for temperatures between -10°C and 65°C. The randomness was measured with a NIST suite of 14 tests and 12-13 tests were consistently passed with a minimum pass rate of 90.6%.

A reversible SW-PUF was also proposed to provide an efficient mechanism for verifiable computations. Reversible SW-PUF is a novel PUF design for computing partial inputs given a set of outputs. We evaluated the reversible SW-PUF in HSPICEk-2015.06 using 45 nm technology. The analysis was performed in terms of two metrics: uniqueness and reliability. The average uniqueness for reverse computation was 30% while for forward computation it was 21%. The reversible software PUF exhibited reliability greater than 99% for temperatures ranging from $-10C$ to $65C$ for both forward and backward computations. Then, we provided an efficient interactive verifiable computation scheme based on the proposed PUF and the Bayesian method. Our approach links outsourced computation with server cloud node hardware to provide proof of correctness of the results with high probability. We evaluated our verifiable computation scheme and demonstrated that it yields faster verification than previous approaches.

Finally, we proposed a privacy-preserving data provenance solution that merges PUF technology with non-interactive zero knowledge proof to provide trustworthy and dependable IoT systems. In our protocol, an IoT device can anonymously send data to an IoT server. The server enrolls an IoT device and verifies all the provenance metadata for data creation and modification. Our scheme can guarantee that the data originated from the specific IoT device that sent it while ensures that the real identity of the owner of the data is not unveiled. The proposed protocol has been evaluated and synthesized with Altera Quartus and implemented on FPGA Altera Cyclone iv. The implementation demonstrates the practicality and feasibility of our solution, while the simulation results achieve practical performance that can be deployed for Resource-Constrained IoT Devices.

5.2 Future Work

The results presented in this work could lead to several future areas of study, including the following:

- Study the threat of machine learning-based software modeling attacks to the SW-PUF and the reversible SW-PUF security, and examine other possible attacks.
- Develop an error-correcting scheme to correct noise in SW-PUF responses.
- Extend an ALU-like PUF to other micro-architecture level units for complete coverage of the instruction set architecture (ISA).
- A program-analysis algorithm could be developed to add more flexibility to the proposed verifiable computation scheme. This would provide a client with the ability to pick the density of the sliced samples to maximize confidence that the server will return the correct results. Such an algorithm can also use an optimization process to pick the ideal set of slices, possibly minimizing the probability of behavior misrepresentation by an adversary.