7-2016

SECURE AND LIGHTWEIGHT HARDWARE AUTHENTICATION USING ISOLATED PHYSICAL UNCLONABLE FUNCTION

Mehrdad Zaker Shahrak
University of Nebraska-Lincoln, zaker.mehrdad@gmail.com

SECURE AND LIGHTWEIGHT HARDWARE AUTHENTICATION USING ISOLATED PHYSICAL UNCLONABLE FUNCTIONS

by

Mehrdad Zaker Shahrak

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Sheng Wei

Lincoln, Nebraska

July, 2016

SECURE AND LIGHTWEIGHT HARDWARE AUTHENTICATION USING ISOLATED PHYSICAL UNCLONABLE FUNCTIONS

Mehrdad Zaker Shahrak, M. S.

University of Nebraska, 2016

Adviser: Sheng Wei

As embedded computers become ubiquitous, mobile and more integrated in connectivity, user dependence on integrated circuits (ICs) increases massively for handling security sensitive tasks as well as processing sensitive information. During this process, hardware authentication is important to prevent unauthorized users or devices from gaining access to secret information. An effective method for hardware authentication is by using physical unclonable function (PUF), which is a hardware design that leverages intrinsic unique physical characteristics of an IC, such as propagation delay, for security authentication in real time. However, PUF is vulnerable to modeling attacks, as one can design an algorithm to imitate PUF functionality at the software level given a sufficient set of challenge-response pairs (CRPs).

To address the problem, we employ hardware isolation primitives (e.g., ARM TrustZone) to protect PUF. The key idea is to physically isolate the system resources that handle security-sensitive information from the regular ones. This technique can be implemented by isolating and strictly controlling any connection between the secure and normal resources. We design and implement a ring oscillator (RO)-based PUF with hardware isolation protection using ARM TrustZone. Our PUF design heavily limits the number of CRPs a potential attacker has access to. Therefore, the modeling attack cannot be performed accurately enough to guess the response of the PUF to a challenge.

Furthermore, we develop and demonstrate a brand new application for the designed PUF, namely multimedia authentication, which is an integral part of multimedia signal processing in many real-time and security sensitive applications. We show that the PUF-based hardware security approach is capable of accomplishing the authentication for both the hardware device and the multimedia stream while introducing minimum overhead.

Finally, we evaluate the hardware-isolated PUF design using a prototype implementation on a Xilinx system on chip (SoC). Particularly, we conduct functional evaluation (i.e., randomness, uniqueness, and correctness), security analysis against modeling attacks, as well as performance and overhead evaluation (i.e., response time and resource usages). Our experimental results on the real hardware demonstrate the high security and low overhead of the PUF in real time authentication.

DEDICATION

To my Father, Dr.Ali Asghar Zaker Shahrak, who always reminded me to believe in my abilities, never afraid of challenges and work hard.
To my Mother, Dr.Mehr Azam Jamali who loved me more than anything in this world. All that I am, or hope to be I owe it to my angel mother.
To my lovely sisters, Dr.Mehrsa Zaker Shahrak and Dr.(to be) Mehrnaz Zaker Shahrak, I might be separated from them by distance, but they are always joined to my heart by love.

ACKNOWLEDGMENTS

I would like to specially thank my advisor, Sheng Wei. I appreciate his guidance and patience while I worked on this thesis. Also, I thank Sheng for supporting me throughout all ups and downs of my thesis. I have learned from him that nobody is perfect. Every one slides here and there, but when they are down, that is not the time to step all over them.

I especially want to thank Jeremiah Goerdt, Mitchel Gerrard and Mikaela Cashman for staying by my side and supporting me through the harshest season of my life.

I want to thank my colleague and friend Mengmei Ye for all of her technical suggestions and her ingenious point of view.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As embedded devices expand in our everyday life and their cost gets lower, there is more demand of having a lightweight authentication method that can be implemented on a prevalent device with little cost [19]. The authentication method should be resistant against interference as the attacker has full physical access to the device most of the time. Furthermore, any authentication procedure adds a processing delay and overhead to the routine performance of the system, which may not meet the requirement of the applications.

One of the approaches of hardware authentication is using a widely distributed public key to encrypt sensitive data and program a secret key into non volatile memory such as EEPROM, and then use cryptographic procedures such as digital signature to authenticate a device [27]. There are several problems with this approach. First, with symmetric key applications, universal devices need to store sensitive information which might be compromised with security attacks. Second, each time an authentication occurs, there is a large number of logical blocks that need to engage in the process of authentication, which makes this procedure vulnerable to side channel attacks as it empties lots of power out of the device. Third, programming device-specific secrets in memories are very expensive since the designer has to keep record of the specific private keys generated for each device.

To address the problems of the secret key-based approach, researchers have proposed physical unclonable function (PUF) as a powerful hardware authentication mechanism. A PUF is a hardware equivalent of a mathematical injective function where it is impossible to reverse engineer the input (challenge) based on the output (response) [5]. The asymmetry PUF creates is one of the most important security feature in real-world: it should be convenient for an authenticated user to use the system while it is impractical for an adversary to counteract it [21]. Therefore, PUF can serve as a biometric signature used for hardware authentication in security sensitive applications.

A PUF is a function in which (1) if we give the same input over and over to it, it replies with the same output repeatedly. (2) Any input results in a unique output. In other words, one can not generate the same output by giving the PUF two different inputs. (3) Changing one bit in input and giving it to PUF, we expect to get a very different output comparing to the previous one. A PUF implementation on hardware is a very useful tool in many ways. First, it presents a way to authenticate a device uniquely. This is useful in applications that need chip authentication or protection of Intellectual Properties (IPs). Second, since PUF is essentially based on randomness, it could be used as a Random Number Generator (RNG). Most importantly, a PUF provides the aforementioned functionalities without storing any secret information on the chip. An ideal PUF relies on specifications of chip itself and, therefore, it is considered impossible to be duplicated [16].

PUF applications can be categorized in three divisions: system authentication, secret key generation, and hardware involved cryptography. For system authentication, PUF can be used for anti-counterfeiting or hardware binding. For secret key generation, PUF can be used for initiating a secret key where the key is embedded in the form of physical characteristics of an IC, such as propagation delay. For hardware cryptography, PUF can be used to generate a digital signature where secret parameters depend on device physical characteristics [15].

Despite many of the security benefits provided by PUF, it is vulnerable to modeling attacks. In a modeling attack, the adversary tries to achieve sufficiently large number of challenge-response pairs (CRPs) to gain a numerical software model on the PUF with lowest error possible so that one can generate the PUF responses without physically possessing the hardware PUF[23]. Another vulnerability of PUF is that one can launch a Denial of Service (DoS) attack on PUF by sending too many challenges to it and blocking the PUF to serve other requests.

To address the aforementioned vulnerability of PUF, the proposed solution in this work is to isolate the input and output of PUF from the outside world and, therefore, it is impossible for the attacker to obtain enough data and derive a model for generating PUF responses to arbitrary challenges. In particular, we embed the hardware PUF in a physically isolated secure zone enabled by hardware isolation technologies, such as ARM TrustZone. Our hardware isolation-based method ensures that the access to challenges and responses of PUF cannot be granted unless it passes a strict security verification. More specifically, we track all the authentication requests and keep the history of the user who issued the requests, as well as the time and number of challenges requested. We reject those requests that ask for too many challenges, or if one user issues requests too frequently in a short period of time.

Furthermore, we develop and demonstrate a brand new application for the designed PUF, namely multimedia authentication, which is an integral part of multimedia signal processing in many real-time and security sensitive applications. We show that the PUF-based hardware security approach is capable of accomplishing the authentication for both the hardware device and the multimedia stream while introducing minimum overhead.

Finally, we evaluate the hardware-isolated PUF design using a prototype implementation on a Xilinx system on chip (SoC). Particularly, we conduct functional evaluation (i.e., randomness and uniqueness) security analysis against modeling attacks, as well as performance and overhead evaluation (i.e., response time and resource usages). Our experimental results on the real hardware demonstrate the high security and low overhead of the PUF in real time authentication.

Chapter 2

Background & Related Work

2.1 PUF Taxonomy

PUFs are physical systems that generate unique and unpredictable responses for given challenges. More precisely, they can be referred to as hardware implementations of secure one-way functions. There are various types of PUFs, each of which has its own security applications and features [23]. The two main categories of PUFs well adopted by the hardware security community are *Strong PUFs* and *Weak PUFs* [21, 5, 4].

2.1.1 Strong PUFs

Strong PUFs are those that have numerous CRPs, so each time the authentication procedure can require new CRPs that have not been used before. As a result, a potential attacker cannot perform a replay attack by recording and applying the CRPs already used in previous authentications.

Although it can prevent replay attacks, the strong PUF is still vulnerable to modeling attacks where the attackers attempt to collect a large number of CRPs and build a software model to emulate the PUF behavior. Typically there is no built-in protection mechanism to restrict the access to responses created by strong PUFs. Also, most electrical strong PUFs operate at low frequencies [14] and, consequently, even short time access to the system enables reading a large number of CRPs.

2.1.2 Weak PUF

Weak PUFs are a special case of saving keys to non-volatile memory. It has only a small number of challenges. The response of a weak PUF is used to derive a cryptographic embedded system and, therefore, the output available to the outside world is an indirect response to the original challenge given to the PUF [23]. Examples of weak PUF include SRAM PUF [11], Butterfly PUF [13], and Coating PUF [29].

One of the advantages of the weak PUF is that it is harder for an adversary to obtain CRPs. Also, the error correction process is done internally with the error correcting helper data stored in non-volatile memory, which ensures security. On the other hand, strong PUFs have the error-correction process handled by external blocks, which have access to the responses.

2.1.3 PUF Examples

The basic concept of the PUF design, which was also introduced in [5][6], is to create a number of delayed path elements and use them randomly to generate random and unique responses. In the following sections we are going to see two examples of PUF designs on FPGA.

2.1.3.1 Ring Oscillator PUF

One of the most lightweight designs that can be implemented effectively on FPGA is Ring Oscillator (RO) PUF (refer to chapter 3). RO leverages delay loops that each

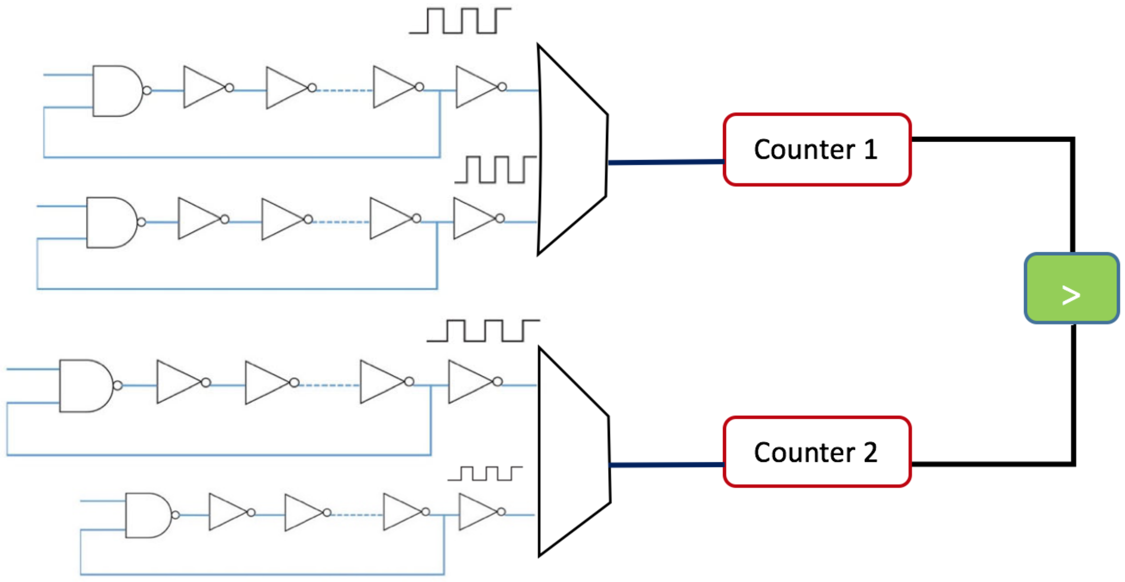


Figure 2.1: Ring Oscillator PUF [27]

oscillates at a specific frequency [27]. Figure 2.1 shows an example of RO PUF. The idea is to make exactly similar paths of ROs and select one of them based on the challenge given to the PUF as input. Then, the selected path will be used as counter clock. Since each path has a unique frequency, the clock works on a unique clock speed and generates the results on a inimitable pace which is leveraged to generate the response based on each challenge. In a RO PUF, all delay loops are identical in theory but, due to the random variations in manufacturing, each loop creates a particular frequency that is slightly different from other loops. Loops are connected to multiplexers where input challenge is being used as selector to connect one input delay loop to a counter. Each multiplexer's output serves as counter clock which in turn propagates a signal to a number of flip flops. As a result, it produces unique response bits to a given input challenge. In this case, the counter produces the results sooner, i.e., the one with faster frequency, is responsible for generating the response.

A RO PUF has lots of benefits comparing to other kinds of PUF. First, RO is

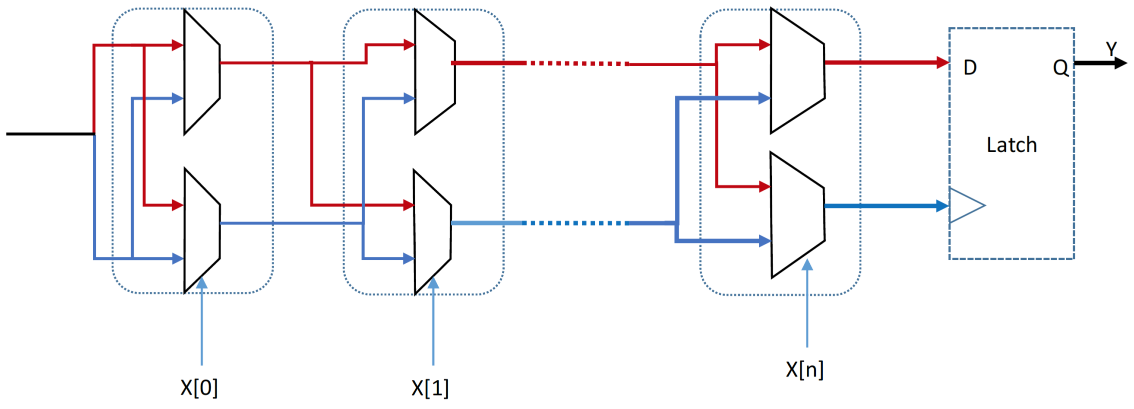


Figure 2.2: Schematic of Arbiter PUF [27]

sensitive to process variations and, as a result, it is extensively used in modeling process variations[18, 20, 26]. Second, the hard-macro technique makes it simpler to design several *identical* RO paths [16].

2.1.3.2 Arbiter PUF

Arbiter PUF uses delay elements in series, where each element is connected to a multiplexer that chooses a path by using the input challenge. In this scenario, the challenge is being used as multiplexers selectors as it is shown in Figure 2.2. The response of the arbiter PUF depends on which path has the fastest signal propagation. The arbiter performs as a referee to tell which path gets to D flip-flop faster and consequently the fastest path will be connected to output Y. Although the Arbiter PUF is fast, it has two main disadvantages. First, it requires symmetrical routing at each stage, otherwise the unity of responses are not guaranteed [17]. This makes the FPGA implementation difficult. Second, Arbiter PUF is prone to modeling attack [23].

Compared to Arbiter PUF, RO PUF is easier to implement on FPGA which is why we chose RO PUF in this thesis for the prototype implementation.

2.2 PUF Threat Models

Although PUF is a powerful hardware security primitive for authentication, it is vulnerable to various attacks. Most of the security attacks against PUF focus on reverse engineering the internal structure of the PUF, so that one can clone and emulate the PUF behavior using a software model [31][23]. Since it is very expensive if not impossible to physically reverse engineer and clone a PUF, most of the existing methods have focused on software model building attacks in the following ways: (1) Collect a sufficiently large set of CRPs and employ machine learning techniques to build a software model for the PUF [23]; or (2) Use side channel signals to obtain information on how the authentication works and build a software emulator to generate the same PUF response [31].

In addition, PUF is vulnerable to denial of service (DoS) attacks, which happens when the attacker succeeds in feeding the inputs of circuit beyond the requests that the PUF can handle. Therefore, the PUF is not only incapable of responding to requests already fed to it, but also cannot handle requests of other users/applications.

There are several PUF protection schemes that have been proposed to prevent model building attacks. Most of the existing approaches focus on how to maximize the randomness of PUF and minimize the correlation between challenges and responses, so that the attackers have to collect an extremely large number of CRPs for the modeling attacks to succeed. For example, XOR-mixed PUF design [27] mixes the responses of arbiter-based PUFs using a certain number of XOR gates in order to obfuscate the PUF design from machine learning-based modeling. Majzoobi et al. design a slender PUF [22] that minimizes the exposure of PUF responses to attackers by conducting substring matching, which significantly increases the difficulty of modeling attack. Controlled PUF design [4] adopts random hashing to the PUF challenges and responses to hide the actual CRPs from the attackers.

The existing PUF security mechanisms provide strong protection against modeling attacks. However, almost all of them require modifications of the existing PUF architecture or the protocol, which not only complicates the implementation and deployment of PUF but also potentially increases the performance overhead. We develop a new hardware isolation-based PUF protection framework to address these issues.

2.3 Hardware Isolation

Hardware isolation techniques, such as ARM TrustZone [1] and Intel SGX [10], create a physically separated runtime environment to protect secret data and other resources on the target system. Since the isolation is implemented at the physical bus level, hardware isolation provides a fundamental security guarantee, which is significantly more robust than software or virtual machine (VM) based solutions. Until now, hardware isolation techniques have been adopted by many software applications, such as protecting the OS kernel [2], language runtime [25], mobile computing [28], and cloud computing [3]. However, it has not been considered in protecting hardware resources. Given the physical level isolation it provides, we regard hardware isolation as a promising method of protecting hardware IP cores, such as the PUF, which are otherwise vulnerable to a variety of attacks. Therefore, in this work, we for the first time develop a hardware isolation-based PUF design for enhanced security.

2.4 Thesis Overview

The rest of this thesis first discusses design specifics and implementation details of RO PUF in Chapter 3, and then describes protecting the design using hardware isolation, namely TrustZone in Chapter 4. Chapter 5 presents a new application of PUF in multimedia authentication. Chapter 6 evaluates the security and performance of the PUF design. Chapter 7 presents our conclusions and describes possible future work.

Chapter 3

Design and Implementation of Ring Oscillator PUF

In this chapter, we present our design of RO PUF on FPGAs. There are different ways to generate the random responses in real hardware, such as using arbiters [6], ring oscillators (ROs) [27], and SRAM [11]. In this work, we employ the RO-based PUF in our design and experiments considering its ease of implementation on FPGAs. However, the hardware isolation-based PUF protection approach does not rely on the specific implementation of PUF and, therefore, other types of PUF designs can be applied as well as long as they conform with the security requirements of PUFs [5].

3.1 Design Elaboration of RO PUF

Our RO PUF is based on the basic design proposed by Suh et al. [27]. To create an 8-bit PUF, we use two 16-input multiplexers, with each taking a 4-bit challenge as the selector[1]. One input from each multiplexer is selected to be used as the clock signal

[1]In this design, while there are maximally 32 ROs that can be connected to the two multiplexers, we use 16 ROs and allow sharing ROs between the two multiplexers to control the area and power

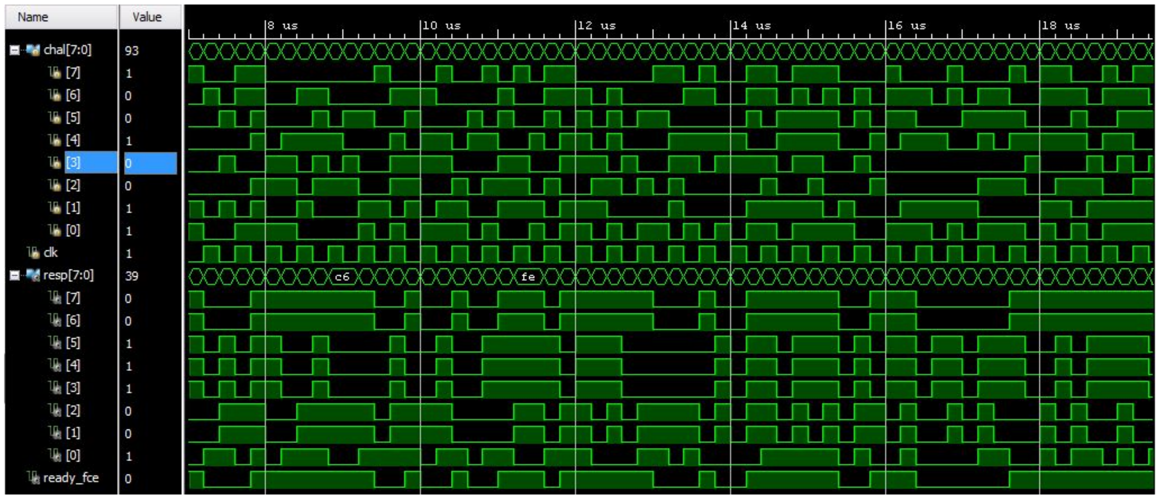


Figure 3.1: Post-Synthesis Timing Simulation

for the counter. Once a valid clock signal is received by the counter, it starts to count with positive edge of its clock until it overflows (i.e., obtaining all 1's in the output and restarts again). Based on the unique frequency of each clock, one of the counters overflows sooner than others, which contributes to and generates the response bits. The result of post synthesis implementation using Xilinx Vivado toolchain is shown in Figure 3.1.

3.2 Addressing The Design Challenges

3.2.1 Solving RO Delay Problem

The main challenge in designing RO PUF is to create identical RO paths and leverage the process variations to make the difference between paths. Since all FPGAs use resource optimization techniques, the similar paths going to be merged into one due to the similarity to save resources of board. This is an unwanted scenario because we

overhead caused by the ROs. Obviously the reuse of ROs may affect the randomness of the PUF. We use the number of ROs as a configurable parameter to control the tradeoff between security and area/power overhead.

want to leverage the intrinsic fabrication variances.

To solve this problem, we create a custom strategy for synthesis, which flattens the hierarchy during LUT mapping (rebuilt) and keeps equivalent registers. This approach solves the problem for implementing the design on real FPGA. However, when simulating the implementation in software, since the software does not reflect the *physical* fabrication differences, we cannot expect to leverage intrinsic process variations. To solve this problem, we hardcode different delays into the RO paths to make them nonidentical for the software simulation phase. To make this nonidentical paths realistic, we use a model of the delay variances due to fabrication technology. For instance, the same transistor standard on different boards have $\pm 5\%$ delay. Therefore, by having the delay value of that standard, we can make different delays in the $\pm 5\%$ range. Table 3.1 shows the the implementation settings in detail.

3.2.2 Design of Multiplexers and Counters

In our PUF design, we use two 16-bit multiplexers, each with a 4-bit selector, which create a PUF with 8-bit CRPs. Also, we use four 4-bit multiplexers to implement each 16-bit multiplexer in the FPGA design.

The counters are both 4-bit counters. Each starts based on its unique clock frequency and stops and generates an output when it overflows. The reason why we select 4-bit counters is that for higher bit counters, it takes longer time to overflow, which causes longer delay and, as a result, the PUF would respond 0 to an arbitrary challenge. On the other hand, for lower bit counters the overflow happens faster than the latches could handle.

3.2.3 Configuring Inner and Outer Clocks

Our RO PUF has two different clocks which need to work and be configured together. The inner clock controls the flip-flops and D-latches outputs, which is the response to the corresponding challenge. The outer clock in turn controls the RO paths being loaded and connected to multiplexers within every positive edge. Therefore, the outer clock should be long enough to cover the frequency of all RO paths. On the other hand, the inner clock should be long enough to let the D-latches transfer their input to the output before they restart, and it should be short enough to clear the latches from all previous values before a new RO path causes a counter to overflow.

As a result, the inner and outer clocks both are responsible for generating the results, and they affect the response time of PUF to a challenge.

3.3 PUF Implementation on SoC

We implement the RO PUF on Xilinx ZedBoard as shown in Figure 3.2, using the Vivado toolchain. The ZedBoard has both programmable logic (PL) and a processing system (PS) with ARM processor. Our PUF design is based on a modified version of the implementation presented in [12]. Figure 3.3 shows our detailed block implementation.

Also, we package the PUF design as a custom Intellectual Property (IP) and connect it to the Xilinx Zynq processing system using an AXI interconnection, as shown in Figure 3.4. Then, a hardware implementation is created from the design and, at the end, the bitstream is generated and downloaded to the board.

The design creates an address map for the ZedBoard processor, which provides necessary addresses to load different registers and memories. To evaluate the PUF performance, we feed the base address of AXI with a PUF challenge, which can be

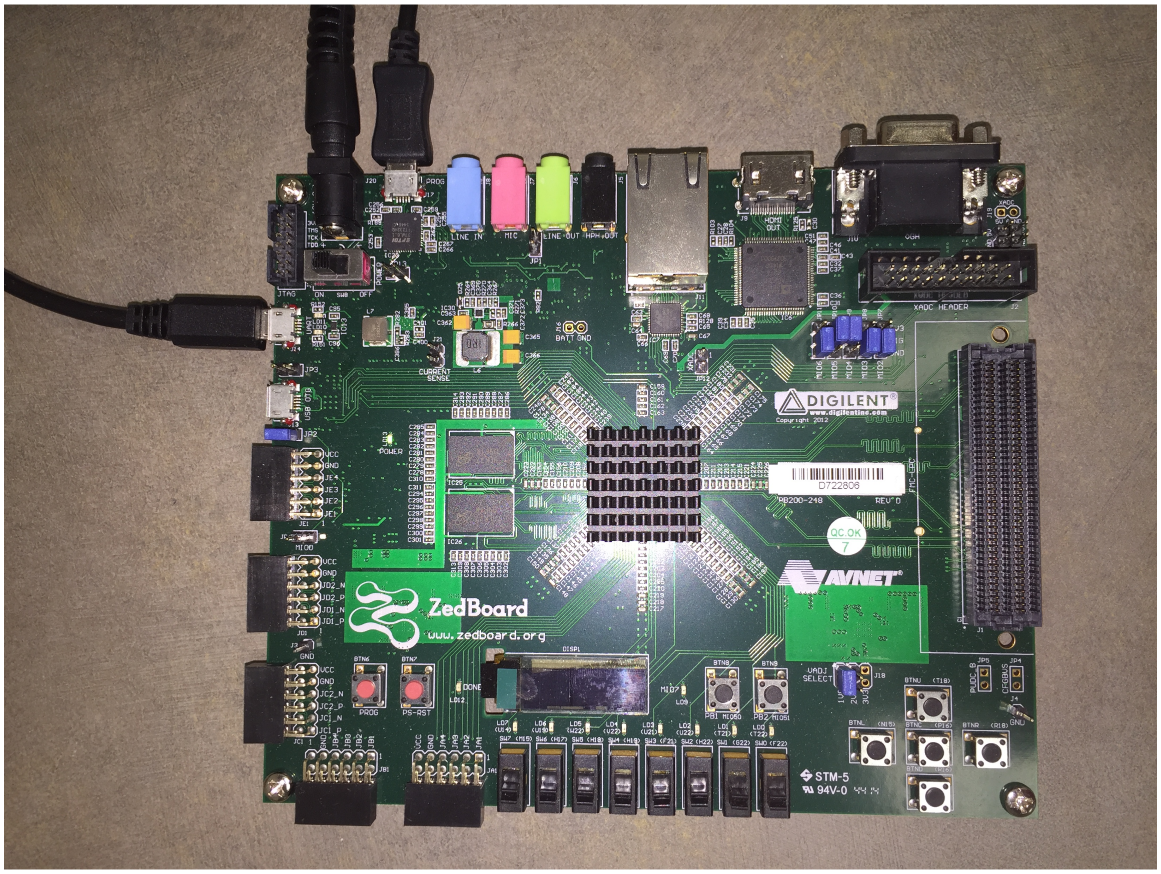


Figure 3.2: Xilinx ZedBoard (Zynq 7000) Development Board

stored into the first slave register of AXI and in turn connects to the PUF input. The response of the PUF is being stored in the second slave register of AXI, which is accessed by the base address plus an offset. Therefore, by printing the second slave register to the terminal, we can see the response of PUF to each corresponding challenge.

By generating the bit stream, the design must pass synthesis and implementation processes. The bitstream is used to program the target FPGA. The generated bit stream is used to be exported to SDK which contains Xilinx device configuration.

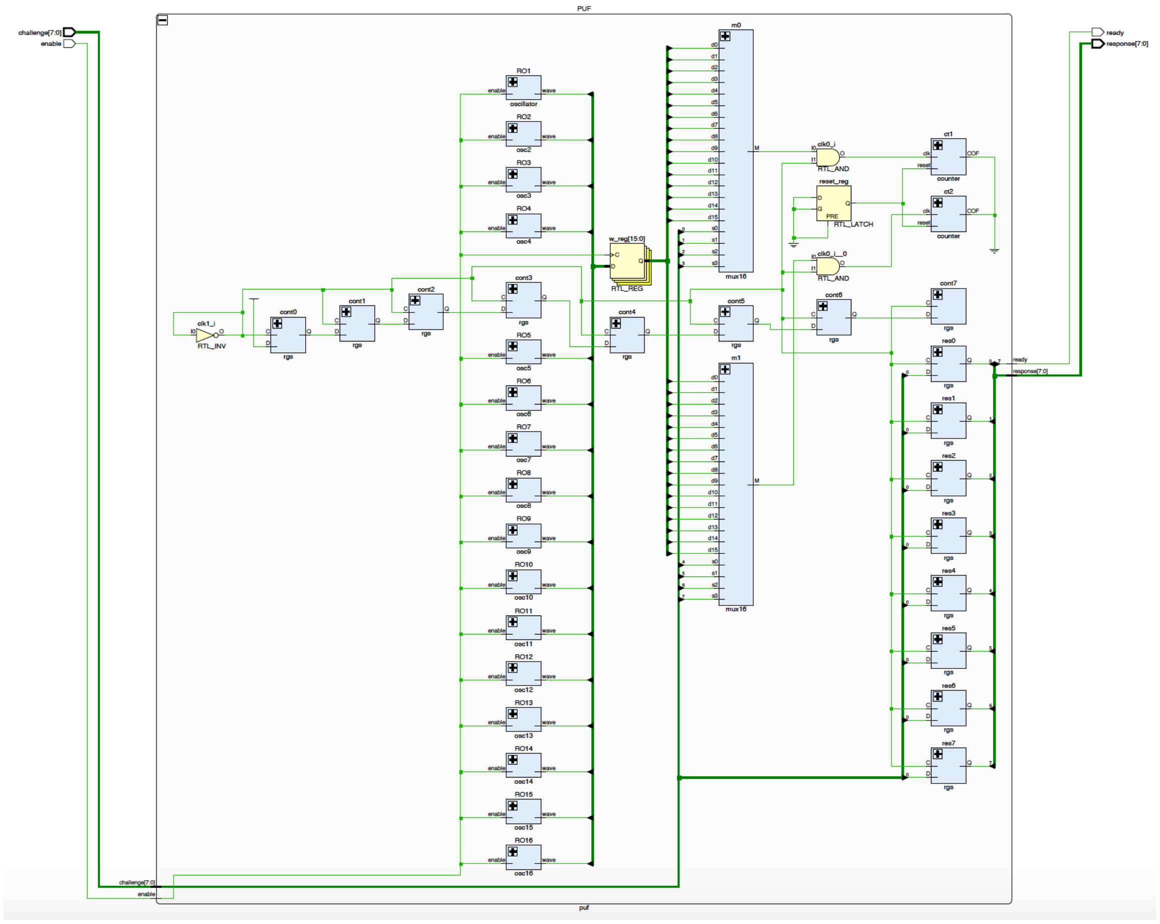


Figure 3.3: Elaborated Design of Ring Oscillator PUF

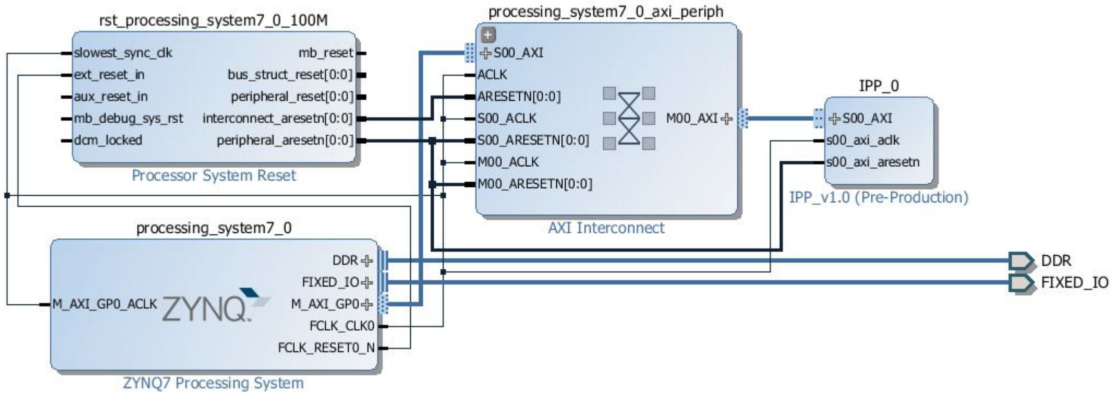


Figure 3.4: Block Diagram and Custom IP Connection Using Xilinx Vivado Toolchain.

Table 3.1: Synthesis Strategy

Name of property	**Mode**
Flatten hierarchy	rebuilt
Keep equivalent registers	checked
Resource sharing	off
Control set opt threshold	0
No LC	checked
More Options	-mode_out_of_context

3.3.1 Settings and Configuration Strategies

To synthesize the PUF design the resource optimization should be turned off. This requires defining custom synthesis and implementation strategies for Xilinx Vivado, as shown in Table 3.1.

As a result of turning off resource optimization, in the simulation phase, there could be some errors generated by Vivado that are due to the creation of the same logic block or path multiple times. To address this simulation issue, we downgrade the level of optimization errors to warnings in Vivado. Therefore, although this solution is against resource optimization, but it continues to simulation phase anyway.

3.3.2 Test Benches

In this study we develop two different groups of test benches. First, we create 5000 random challenges and feed them to the design, from which we receive 5000 pairs of responses. Second, we create 5000 pairs of random challenges (10000 challenges total), where in each pair there is only one bit difference ranging from bit 1 to bit 8 (in a 8 bit CRP) randomly.

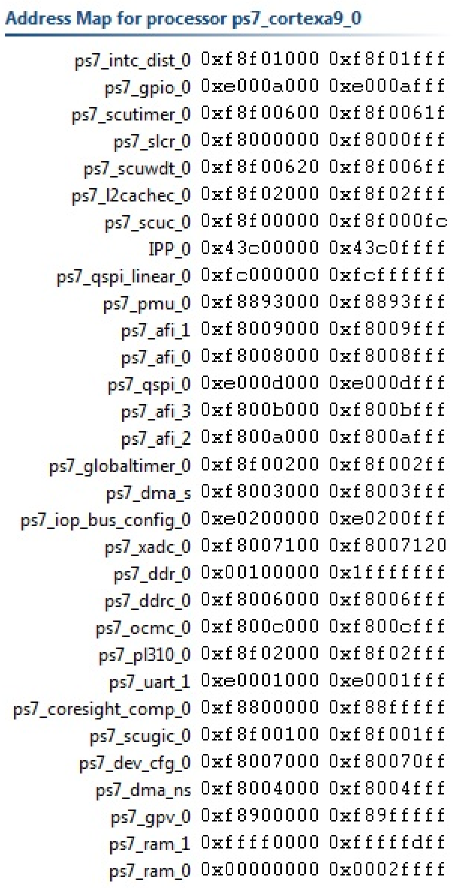
For the first test bench, we will check the probability distribution of 0 and 1 in each position of response. The expected result is that the distribution should be very similar to the distribution of challenges. In that case, we can conclude that the

PUF is random and the probability of each combination to happen is equal to other combinations (in this case the probability of a combination to happen is $(\frac{1}{2})^8$). Also, we checked the randomness of this test set using NIST statistical package.

For the second test bench, we aim to check whether our design is immune to modeling attacks. In this case, the testing scenario is to change one bit, feed it to PUF and compare both of CRPs. If the response after one bit change is too much different (half of the challenge length or more) it means that the PUF functionality is too complex to be imitated and, therefore, the proposed PUF design cannot be easily modeled.

3.3.3 ZedBoard Programming

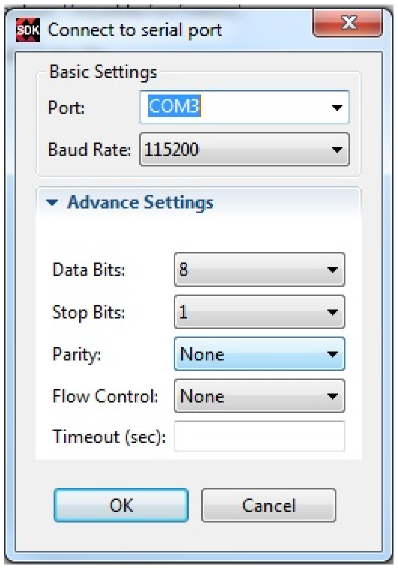
To program the board we must load the address map and the configurations created by generating the bitstream to the target board. An example of address map shown in Figure 3.5. We use the standard JTAG interface to program and debug the ZebBoard. Then, we connect Xilinx SDK to the UART port of the board to observe the output (responses). The settings to connect to the UART are shown in Figure 3.6.



Address Map for processor ps7_cortexa9_0

ps7_intc_dist_0	0xf8f01000	0xf8f01fff
ps7_gpio_0	0xe000a000	0xe000afff
ps7_scutimer_0	0xf8f00600	0xf8f0061f
ps7_slcr_0	0xf8000000	0xf8000fff
ps7_scuwdt_0	0xf8f00620	0xf8f006ff
ps7_l2cachec_0	0xf8f02000	0xf8f02fff
ps7_scuc_0	0xf8f00000	0xf8f000fc
IPP_0	0x43c00000	0x43c0ffff
ps7_qspi_linear_0	0xfc000000	0xfcffffff
ps7_pmu_0	0xf8893000	0xf8893fff
ps7_afi_1	0xf8009000	0xf8009fff
ps7_afi_0	0xf8008000	0xf8008fff
ps7_qspi_0	0xe000d000	0xe000dfff
ps7_afi_3	0xf800b000	0xf800bfff
ps7_afi_2	0xf800a000	0xf800afff
ps7_globaltimer_0	0xf8f00200	0xf8f002ff
ps7_dma_s	0xf8003000	0xf8003fff
ps7_iop_bus_config_0	0xe0200000	0xe0200fff
ps7_xadc_0	0xf8007100	0xf8007120
ps7_ddr_0	0x00100000	0x1fffffff
ps7_ddrc_0	0xf8006000	0xf8006fff
ps7_ocmc_0	0xf800c000	0xf800cfff
ps7_pl310_0	0xf8f02000	0xf8f02fff
ps7_uart_1	0xe0001000	0xe0001fff
ps7_coresight_comp_0	0xf8800000	0xf88fffff
ps7_scugic_0	0xf8f00100	0xf8f001ff
ps7_dev_cfg_0	0xf8007000	0xf80070ff
ps7_dma_ns	0xf8004000	0xf8004fff
ps7_gpv_0	0xf8900000	0xf89fffff
ps7_ram_1	0xffff0000	0xfffffdff
ps7_ram_0	0x00000000	0x0002ffff

Figure 3.5: Adress Map Created for ARM Cortex 9 Processor



Connect to serial port

Basic Settings

Port: COM3

Baud Rate: 115200

Advance Settings

Data Bits: 8

Stop Bits: 1

Parity: None

Flow Control: None

Timeout (sec):

Figure 3.6: Connection Settings to Serial Port of Zedboard

Chapter 4

Protecting PUF using Hardware Isolation

In this chapter, we present our hardware isolation-based approach to protect PUF from modeling attacks. As we introduced in Chapter 2, hardware isolation techniques provide fundamental protection to the system resources by creating a physically isolated runtime environment. We leverage the security feature of hardware isolation and, in particular, ARM TrustZone [1] to isolate and strictly control the access to the PUF. Therefore, it prevents the adversaries from collecting sufficient CRPs at their choice to issue modeling attacks against the PUF.

Figure 4.1 shows the overall framework of our hardware isolation-based PUF protection scheme. ARM TrustZone divides the application runtime into a normal world (NW) and a secure world (SW), where SW has direct access to resources in NW, but NW cannot directly access SW. Instead, any access to the SW must go through the secure monitor in the monitor mode via a secure monitor call (SMC). The secure monitor determines whether to switch the CPU mode to SW and grant access to the resources in SW by following a access control policy.

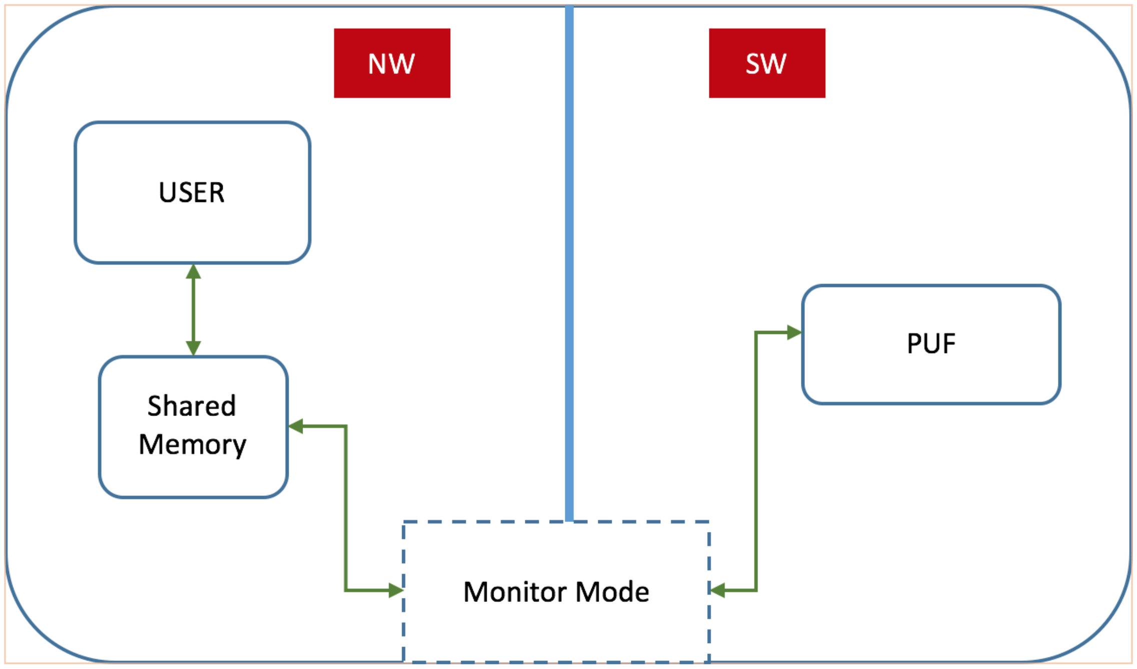


Figure 4.1: Using TrustZone to Protect PUF

To protect our PUF design, we deploy the PUF as an IP core in the SW and, therefore, the access to PUF is under the protection of the hardware isolation framework. Algorithm 1 shows the authentication process in our design enabled by the hardware isolation, which occurs once the secure monitor receives an SMC from the NW application requesting PUF responses.

Since the system keeps track of users who demand PUF authentication, if the same user issues too many requests (e.g., beyond a threshold T), the authentication process will reject its requests. By restricting access to PUF CRPs, a potential attacker can not get data to train the learning algorithm to imitate PUF. Also, this restriction prevents the attacker from successfully launches Denial of Service (DoS) attack.

Algorithm 1 Device authentication using PUF protected in TrustZone

```
Create-Memory(M)
Create-Struct(H)
The user in NW side asks for authentication process
$M \leftarrow K$ Challenges
Secure-Monitor-Mode()
Monitor-Call()
H (user ID) = Struct-Store(time,$K$)
if $0 < K \leq T$ then
    SW takes the next processing cycle
    while SW has control do
        SW access M
        $PUF \leftarrow K$
        Shared Memory $\leftarrow$ PUF
        if $K$ responses generated then
            Secure-Monitor-Mode()
            Monitor-Call()
        end if
    end while
end if
NW takes the next processing cycle
```

Chapter 5

Two-Way Real Time Multimedia Stream Authentication Using Physical Unclonable Functions

In this chapter, we present a novel application of PUF, namely multimedia stream authentication, which is an integral part of multimedia signal processing in many real-time and security sensitive applications, such as video surveillance. We show that PUF-based security primitive is not only capable of accomplishing the authentication for both the hardware device and the multimedia stream but, more importantly, introduce minimum performance, resource, and power overhead.

5.1 Overview of Multimedia Authentication

Real time multimedia streaming systems, such as video surveillance systems, are gaining an increasing level of applications especially in the era of fast evolving internet of things (IoTs). For example, more and more video surveillance systems have been installed to monitor the real time security status of private residences, banks, highway traffic, etc. Since many of these systems are deployed for remote security monitoring, the video must be streamed to the monitoring portal (i.e., the remote receiving end) via network communications. In this case, the security and integrity of the multimedia stream become critical.

There are typically two potential threats that may compromise the surveillance video stream. First, it is possible for an adversary to switch the source of the stream or partially modify the video stream content before it is delivered to the receiver. Second, an unauthorized viewer may receive and watch the video stream that he/she is not supposed to view. The first threat may seriously compromise the security and thus the entire purpose of deploying such a surveillance system, and the second threat may raise privacy concerns for the objects being monitored.

The aforementioned threats raise two design goals for real time multimedia authentication, which we focus on in this chaper: (1) *Multimedia Authentication*, which verifies whether the multimedia content could have been modified before playback at the receiver end; and (2) *Device Authentication*, which examines whether the receiver has the permission to play the streamed multimedia content. While both threats can be well addressed by employing a full-fledged video digital rights management (DRM) scheme widely used by Internet entertainment video streaming [30], it requires video encryption and thus sophisticated license and key management strategies that are too heavy weight for the real time video surveillance.

Therefore, most of the existing approaches have focused on non-encryption and non-DRM based video authentication solutions, such as signature-based [9][7][8] approaches, where the sender signs the stream using a public key and the receiver verifies it using the corresponding private key. There are many research efforts that focus on how to reduce the overhead of signing and verifying the stream by using a variety of ways for hash chaining [7][8][33][32]. However, the existing multimedia authentication approaches appear to be on another extreme compared to full DRM, as they lack the support of device authentication and thus have no control over who can view the content. This situation may be fine in a small constrained network. However, with the widespread IoT deployment, privacy issues have drawn significantly more attention than ever before, and it becomes necessary to deploy a lightweight device or user authentication scheme for the video surveillance scenario. In addition, the performance of the signature and the overhead of the authentication information exchange can still be improved.

We develop a two-way real time multimedia authentication scheme that covers both device and multimedia content authentications. As different from the existing approaches, our idea is to employ PUF to achieve the authentication goals. In particular, we employ the unique CRPs corresponding to the hardware PUF to authenticate the receiver device. Also, we adopt the hardware accelerated one-way function enabled by PUF to improve the existing hash chaining-based method in multimedia authentication. To the best of our knowledge, this is the first use case of hardware security primitives in real time multimedia authentication, and we expect the current and the continuing work could provide a new direction for improving the quality of experience (QoE) in multimedia authentication. Real time multimedia streaming systems, such as video surveillance systems, are gaining an increasing level of applications especially in the era of fast evolving internet of things (IoTs). For example, more and more video surveillance systems have been installed to monitor the real time security status of private residences, banks, highway traffic, etc. Since many of these systems are deployed for remote security monitoring, the video must be streamed to the monitoring portal (i.e., the remote receiving end) via network communications. In this case, the security and integrity of the multimedia stream become critical.

5.2 PUF-based Multimedia Authentication Protocol

In this section, we discuss the details of the multimedia authentication protocol based on the PUF, which authenticates both the device and the multimedia stream in a surveillance video streaming scenario.

5.2.1 Device Authentication

In device authentication our goal is to let the sender determine whether the receiver device, which is requesting the video stream, is a registered device that has permission to view the surveillance video. Figure 5.1 illustrates the authentication protocol to achieve the goal leveraging PUFs. There is one PUF involved for each pair of sender and receiver, and the PUF is physically possessed by the receiver. The sender hosts a secure database that contains a sufficient set of CRPs corresponding to the receiver's PUF. There are several ways in which the sender could initialize the CRP database. For example, the sender can collect a large set of CRPs from the actual PUF before delivering it to the receiver, or the sender can maintain a software model of the PUF, obtained from the manufacturer, and generate the CRP in the real time. For the discussion of the device authentication protocol, we do not differentiate these detailed CRP generation mechanisms.

In an authentication session, the sender randomly picks K CRPs from the database, where K is a parameter determined by the sender for different levels of security, and sends only the challenges to the receiver. Upon receiving the challenges, the receiver applies them one by one to the hardware PUF, collects the responses, and sends them back to the sender. Then, the sender verifies if the received responses match with the

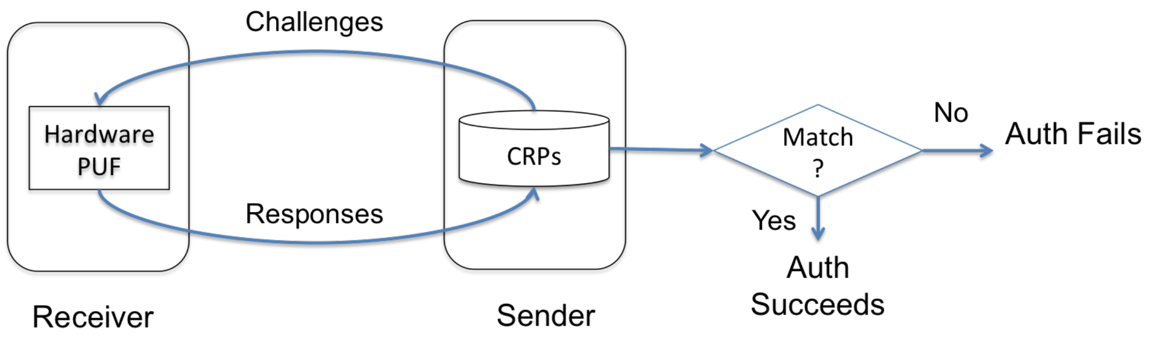


Figure 5.1: Flow of The Device Authentication Protocol.

records in the database. If they do, the sender confirms that the device is authenticated and starts the video stream. Otherwise, the sender will claim an authentication failure and do not initiate the stream.

5.2.2 Multimedia Stream Authentication

For multimedia content authentication, i.e., verifying the integrity of the stream, we adopt the hash chaining method proposed by Gennaro and Rohatgi [7] with our improvement based on PUF. We first split the video stream into continuous streams, which is compliant with the existing HTTP video streaming mechanisms. Then, for each segment i, [7] proposed to embed a one-time public key and the one-time signature of itself with respect to the key contained in segment $i-1$. In this way, there will be a chained signing and verifying procedure as shown in Figure 5.2.

Although the chaining-based signature scheme eliminates the need of the whole stream to verify the signature, it requires storing the one-time public key in each segment, which increases the segment sizes as well as the complexity for processing. In our PUF-based scheme, since both the sender and receiver holds either a software model or hardware PUF, they can both use the PUF to generate the same public key (e.g., using a hash or a portion of the segment as input challenges) and do not need

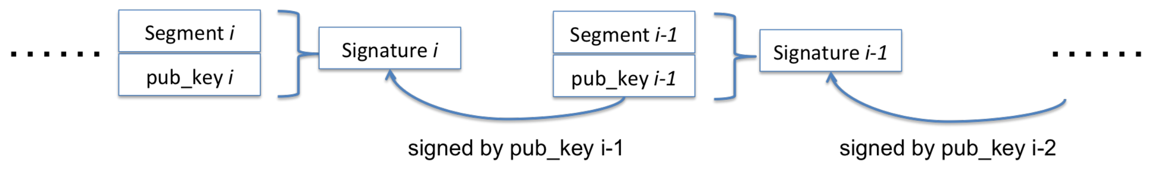


Figure 5.2: Flow of The Chaining-based Signature Scheme for Multimedia Authentication [7].

to exchange the key at the runtime.

Chapter 6

Experimental Results

In this chapter, we evaluate the hardware-isolated PUF design in terms of functionality, security, and performance. Our experiments are based on the real hardware implementation of the prototype using Xilinx Zedboard, as we presented in Chapter 4.

6.1 Security Analysis of the Isolated PUF

The fundamental security of the isolated PUF is guaranteed by the hardware isolation technique being used, namely the ARM TrustZone [1], which ensures that the normal world does not have direct access to the secure world. Also, the isolated PUF is able to prevent the attackers from obtaining enough data to successfully launch a modeling attack, as there is a strict access control procedure deployed in the secure world to handle world switching requests. Furthermore, one cannot mount a DoS attack as the additional requests will be rejected if the total number of requests goes beyond the predefined threshold, as enforced by the access control mechanism.

Table 6.1: NIST Statistical Test

Name of Test	Test No.	P-value Results
Frequency (Monobit) Test	1	0.200590
Frequency Test within a Block	2	0.098598
Runs Test	3	0.918913
Test for the Longest Run of Ones in a Block	4	0.679063
Binary Matrix Rank Test	5	0.024298
Discrete Fourier Transform (Spectral) Test	6	0.588255
Non-overlapping Template Matching Test	7	0.655009
Overlapping Template Matching Test	8	0.272950
Maurer's Universal Statistical Test	9	0.076909
Linear Complexity Test	10	0.904980
Serial Test	11	0.481101
Approximate Entropy Test	12	0.341812
Cumulative Sums (Cusum) Test	13	0.296968
Random Excursions Test	14	0.895151
Random Excursions Variant Test	15	0.8665030

6.2 Functional Evaluation

6.2.1 Randomness Test

We evaluate the randomness of the PUF responses using the NIST statistical test suite [24], which is a commonly used benchmark for entropy analysis. In particular, we generate 5000 CRPs from the PUF and run the 15 different NIST tests listed in Table 6.1. According to NIST, a test is considered as passed if the P-value is greater than 0.01. As shown in Table 6.1, the results obtained from our PUF experiments pass all the tests, which indicates that the generated responses closely resemble a randomly generated set.

We also evaluate the possibilities of 0s and 1s in each bit of the response for the 5000 CRPs. Ideally, the probabilities of 0s and 1s should be 50%. Figure 6.1 shows the probability of 0s in our evaluation, which is close to the ideal results.

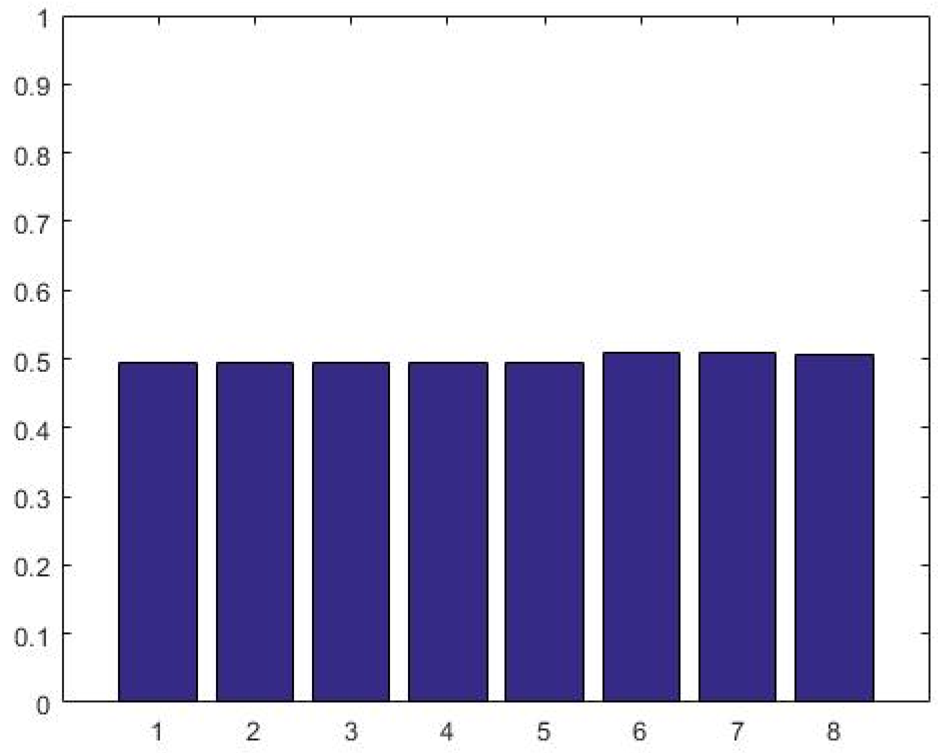


Figure 6.1: Probability of 0s in Each Position of The Response

6.2.2 Hamming Distance Test

We further evaluate the PUF design using hamming distance test. We give 5000 pairs of randomly generated challenges to a single PUF, where each pair has only 1 bit of difference. For an ideal PUF, we expect to obtain a hamming distance that is 50% of the total length of the response given the 1-bit difference in the challenges. For example, the ideal hamming distance for an 8-bit PUF is 4 bits in such a test. This creates the most challenging situation for an attacker to build an analytical model and attempt to guess the responses without physically possessing the PUF hardware. Figure 6.2 shows the frequency distribution of the hamming distances resulted from the 5000 CRPs. We observe that the most frequent hamming distances are 4, 3, and

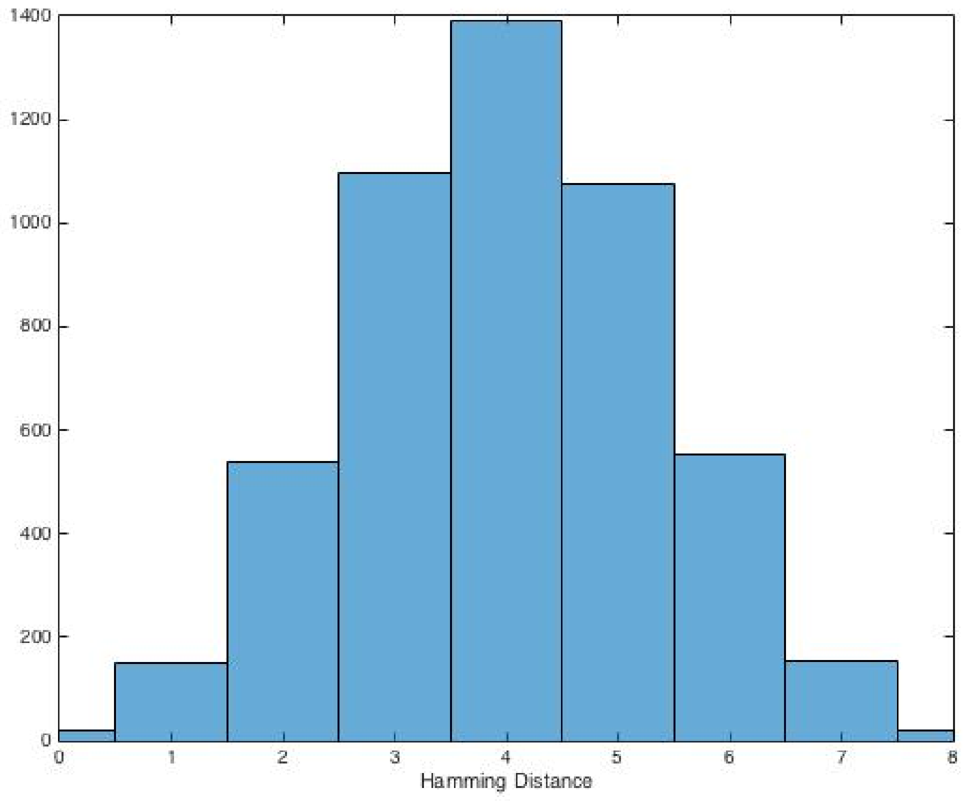


Figure 6.2: Histogram of Hamming Distance Between Pairs

5, which match with our expectation. In particular, we have 99.6% different responses once one bit is changed in the input challenge, and only 0.4% of responses are not different due to the one bit change in input.

6.3 Performance & Overhead Evaluation

We evaluate the performance of the isolated PUF by measuring its response delay between it receives the input challenge and it generates the output response. To accommodate for the random noise and possibly different delays in processing different input challenges, we run 10 groups of timing experiments using random sets of challenges, the sizes of which range from 10 to 100. Figure 6.3 shows the timing evaluation results comparing the isolated PUF with the regular PUF without hardware isolation[1]. We observe that the isolated PUF demonstrates only slight performance overhead as compared to the regular PUF, which is due to the world switching delay and the load/store operations to access the shared memory.

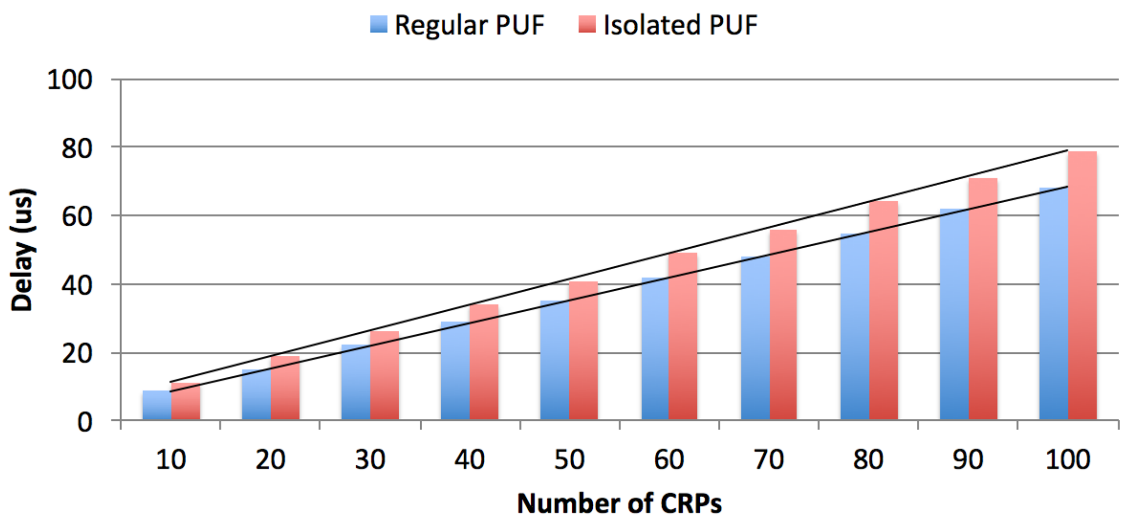


Figure 6.3: PUF Timing Evaluation Results

Furthermore, Table 6.2 shows the resource usage of the PUF implementation obtained from the simulation report using the Xilinx Vivado toolchain. We observe that the PUF uses very small portion of resource, i.e. less than 5% utilization for all resources on the ZedBoard.

[1]The timing measurement of the isolated PUF does not include the delay caused by security verification, as it is customizable based on the security requirements

Table 6.2: Resource Usage of The Isolated PUF Design

Resource Type	Used	Available	Utilization
Slice LUTs	530	53200	1.00%
LUT as Logic	462	53200	0.87%
LUT as Memory	68	17400	0.39%
LUT Flip Flop Pairs	728	53200	1.37%
Slice Register	789	106400	0.74%
Register as Flip Flop	789	106400	0.74%
BUFGCTRL	1	32	3.13%

Chapter 7

Conclusions and Future Work

In this study, we have developed a hardware isolation-based approach to protect PUFs from security attacks. In particular, we embedded the target PUF in a physically isolated secure environment, as enabled by the ARM TrustZone technology, which prevents the attacker's direct access to the PUF. Also, we deployed an access control policy to prevent unauthorized or excessive requests to the PUF, to prevent modeling and DoS attacks. Furthermore, we showed that PUF as a hardware security primitive can be used in a multimedia streaming application to provide both device and multimedia content authentications. Our system implementation and experiments on Xilinx Zedboard verified the security and performance of the proposed approach.

For the future work, it is very interesting to look into the FPGA fatigue patterns as it changes the paths of design in passage of time. Therefore, the responses to challenges might be different from what we currently observe. Also, it will be interesting to conduct a more intensive study and evaluation on the resilience of PUF against a broader set of threat models.

Bibliography

[1] ARM security technology: Building a secure system using TrustZone technology. `http://infocenter.arm.com/\\help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html`. Acessed: 2016-04-23.

[2] A. Azab, K. Swidowski, R. Bhutkar, J. Ma, W. Shen, R. Wang, and P. Ning. SKEE: A lightweight secure kernel-level execution environment for ARM. *The Network and Distributed System Security Symposium (NDSS)*, 2016.

[3] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. Abu Ghazaleh, and R. Riley. Iso-X: A flexible architecture for hardware-managed isolated execution. pages 190–202, 2014.

[4] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Controlled physical random functions. *Computer Security Applications Conference*, pages 149–160, 2002.

[5] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Silicon physical random functions. *ACM Conference on Computer and Communications Security (CCS)*, pages 148–160, 2002.

[6] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Delay-based circuit authentication and applications. *ACM symposium on Applied computing (AC)*, pages 294–301, 2003.

[7] R. Gennaro and P. Rohatgi. How to sign digital streams. *Annual International Cryptology Conference (CRYPTO)*, pages 180–197, 1997.

[8] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. *Network and Distributed Systems Security Symposium (NDSS)*, pages 13–22, 2001.

[9] M. Hefeeda and K. Mokhtarian. Authentication schemes for multimedia streams: Quantitative analysis and comparison. 6(1):6:1–6:24, 2010.

[10] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. D. Cuvillo. Using innovative instructions to create trustworthy software solutions. *International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, Article 11, 2013.

[11] D.E. Holcomb, W.P. Burleson, and K. Fu. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. volume 7, 2007.

[12] I. Jimenez. PUF implementation. `https://spaces.usu.edu/download/attachments/53052062/IvanJimenezPUFReport.pdf?version=1`. Acessed: 2016-05-11.

[13] J. Guajardo Maes R. Schrijen G.J. Kumar, S.S. and P. Tuyls. The butterfly puf protecting ip on every fpga. *Hardware-Oriented Security and Trust. HOST. IEEE International Workshop on*, pages 67–70, 2008.

[14] J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. Van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. *IEEE VLSI Circuits Symposium*, pages 176–179, 2004.

[15] R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. *Towards Hardware-Intrinsic Security (THIS)*, pages 3–37, 2010.

[16] A. Maiti and P. Schaumont. Improved ring oscillator PUF: an FPGA-friendly secure primitive. In *Journal of cryptology*, volume 24, pages 375–397, 2011.

[17] S. Morozov, A. Maiti, and P. Schaumont. An analysis of delay based PUF implementations on FPGA. *Reconfigurable Computing: Architectures, Tools and Applications*, pages 382–387, 2010.

[18] H. Onodera. Variability: Modeling and its impact on design. In *IEICE Transactions on Electronics*, volume 89, pages 342–348, 2006.

[19] E. Ozturk, G. Hammouri, and B. Sunar. Towards robust low cost authentication for pervasive devices. *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 170–178, 2008.

[20] L.T. Pang and B. Nikolic. Measurements and analysis of process variability in 90 nm CMOS. In *IEEE Journal of Solid-State Circuits*, volume 44, pages 1655–1663, 2009.

[21] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. In *American Association for the Advancement of Science (AAAS)*, volume 297, pages 2026–2030, 2002.

[22] M. Rostami, M. Majzoobi, F. Koushanfar, D. Wallach, and S. Devadas. Robust and reverse-engineering resilient PUF authentication and key-exchange by substring matching. In *IEEE Transactions on Emerging Topics in Computing (ITETC)*, volume 2, pages 37–49, 2014.

[23] U. Ruhrmair, F. Sehnke, J. Solter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. *ACM Conference on Computer and Communications Security (CCS)*, pages 237–249, 2010.

[24] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. `http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf`. Acessed: 2016-04-23.

[25] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM TrustZone to build a trusted language runtime for mobile applications. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 67–80, 2014.

[26] P. Sedcole and P. Cheung. Within-die delay variability in 90nm FPGAs and beyond. *IEEE International Conference on Field Programmable Technology (FPT)*, pages 97–104, 2006.

[27] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference (DAC)*, pages 9–14, 2007.

[28] H. Sun, K. Sun, Y. Wang, and J. Jing. TrustOTP: Transforming smartphones into secure one-time password tokens. In *ACM Conference on Computer and Communications Security (CCS)*, pages 976–988, 2015.

[29] P. Tuyls, G.J. Schrijen, B. kori, J. Van Geloven, N. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 369–383, 2006.

[30] R. Wang, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. Steal this movie: Automatically bypassing drm protection in streaming media services. *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 687–702, 2013.

[31] S. Wei, J. Wendt, A. Nahapetian, and M. Potkonjak. Reverse engineering and prevention techniques for physical unclonable functions using side channels. *Design Automation Conference (DAC)*, pages 1–6, 2014.

[32] C. Wong and S. Lam. Digital signatures for flows and multicasts. In *IEEE/ACM Transactions on Networks*, volume 7, pages 502–513, 1999.

[33] Z. Zhang, Q. Sun, W. Wong, and A. Proposal. of butterfly-graph based stream authentication over lossy networks. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 784–787, 2005.