RICE UNIVERSITY

Lightweight Silicon-based Security Concept, Implementations, and Protocols

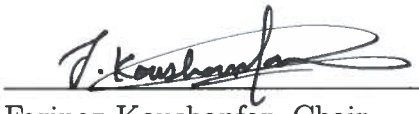
by

Mehrdad Majzoobi

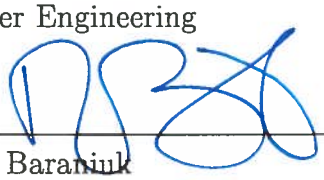
A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:



Farinaz Koushanfar, Chair
Associate Professor of Electrical and
Computer Engineering



Richard Baraniuk
Victor E. Cameron Professor of Electrical
and Computer Engineering

Dan Wallach
Professor of Computer Science

Houston, Texas

March, 2012

ABSTRACT

Lightweight Silicon-based Security
Concept, Implementations, and Protocols

by

Mehrdad Majzoobi

Advancement in cryptography over the past few decades has enabled a spectrum of security mechanisms and protocols for many applications. Despite the algorithmic security of classic cryptography, there are limitations in application and implementation of standard security methods in ultra-low energy and resource constrained systems. In addition, implementations of standard cryptographic methods can be prone to physical attacks that involve hardware level invasive or non-invasive attacks.

Physical unclonable functions (PUFs) provide a complimentary security paradigm for a number of application spaces where classic cryptography has shown to be inefficient or inadequate for the above reasons. PUFs rely on intrinsic device-dependent physical variation at the microscopic scale. Physical variation results from imperfections and random fluctuations during the manufacturing process which impact each device's characteristics in a unique way. PUFs at the circuit level amplify and capture variation in electrical characteristics to derive and establish a unique device-dependent challenge-response mapping.

Prior to this work, PUF implementations were unsuitable for low power applications and vulnerable to wide range of security attacks. This doctoral thesis presents a coherent framework to derive formal requirements to design architectures and proto-

cols for PUFs. To the best of our knowledge, this is the first comprehensive work that introduces and integrates these pieces together. The contributions include an introduction of structural requirements and metrics to classify and evaluate PUFs, design of novel architectures to fulfill these requirements, implementation and evaluation of the proposed architectures, and integration into real-world security protocols.

First, I formally define and derive a new set of fundamental requirements and properties for PUFs. This work is the first attempt to provide structural requirements and guideline for design of PUF architectures. Moreover, a suite of statistical properties of PUF responses and metrics are introduced to evaluate PUFs.

Second, using the proposed requirements, new and efficient PUF architectures are designed and implemented on both analog and digital platforms. In this work, the most power efficient and smallest PUF known to date is designed and implemented on ASICs that exploits analog variation in sub-threshold leakage currents of MOS devices. On the digital platform, the first successful implementation of Arbiter-PUF on FPGA was accomplished in this work after years of unsuccessful attempts by the research community. I introduced a programmable delay tuning mechanism with pico-second resolution which serves as a key component in implementation of the Arbiter-PUF on FPGA. Full performance analysis and comparison is carried out through comprehensive device simulations as well as measurements performed on a population of FPGA devices.

Finally, I present the design of low-overhead and secure protocols using PUFs for integration in lightweight identification and authentication applications. The new protocols are designed with elegant simplicity to avoid the use of heavy hash operations or any error correction. The first protocol uses a time bound on the authentication process while second uses a pattern-matching index-based method to thwart reverse-engineering and machine learning attacks. Using machine learning methods during the commissioning phase, a compact representation of PUF is derived and stored in a database for authentication.

“Dedicated to my parents”

Contents

Abstract ii

List of Illustrations viii

List of Tables xv

1 Introduction 1

1.1 Focus 4

1.2 Thesis Organization 7

2 Background 8

3 Related Literature 16

3.1 Vulnerability analysis and countermeasures 18

3.2 Hardware True Random Number Generation 20

4 PUFs based on timing variations 23

4.1 Delay Signature Extraction 23

4.1.1 Signature extraction system 27

4.1.2 Characterization accuracy 29

4.1.3 Parameter extraction 30

4.2 Timing PUF 31

4.2.1 Pulse challenge 32

4.2.2 Binary challenge 33

4.2.3 Placement challenge 34

4.3 Response robustness 35

4.3.1 Linear Calibration 36
4.3.2 Differential Structure 38
4.4 Experimental evaluations 41
4.5 Arbiter PUF on FPGA 49
4.5.1 Tuning with Programmable Delay Lines 51
4.5.2 PDL-based Symmetric Switch 52
4.6 Precision Arbiter 54
4.7 Robust responses 55
4.8 Robustness versus Entropy 57
4.9 Experimental Evaluation 58
4.9.1 Programmable delay line 58
4.9.2 Arbiter-based PUF evaluation 60
4.9.3 Measurement setup 60
4.9.4 Tuning the PUF 61
4.9.5 Majority Voting 62
4.9.6 Robust response classification 63
4.9.7 Robustness versus entropy 64
4.9.8 Correlation between effects of temperature and power supply variations 65

5 PUFs based on current variations 74
5.1 Concept and circuit realization 74
5.2 Experimental results 78

6 Authentication Protocols 86
6.1 Authentication Protocol 86
6.1.1 Classic Authentication 86
6.1.2 Time-bounded Authentication Using Reconfigurability 87
6.2 Attacks and Countermeasures 90

6.3 Slender PUF Protocol 91
6.3.1 Slender PUF protocol steps 93
6.3.2 Secret sharing 95
6.4 Analysis of attacks 96
6.4.1 PUF modeling attack 96
6.4.2 Random guessing attack 99
6.4.3 Compromising the random seed 100
6.4.4 Substring replay attack 101
6.4.5 Exploiting non-idealities of PRNG and PUF 102
6.5 Experimental evaluation 104
6.6 Hardware implementation 112

7 Conclusion **117**

A TRNG **119**
A.1 TRNG System Design 119
A.2 Experimental results 126

B Plots **134**

Bibliography **160**

Illustrations

2.1 The conceptual architecture of Strong PUF. 10
2.2 Arbiter-based PUF introduced in [1]. 11
2.3 Optional caption for list of figures 13
2.4 Optional caption for list of figures 15

3.1 Delay-based PUF. 17
3.2 TRNG based on sampling the ring oscillator phase jitter. 21

4.1 The timing signature extraction circuit. 24
4.2 The probability of observing timing failure as a function of clock pulse width, T. 27
4.3 The architecture for chip level delay extraction of logic components. . 28
4.4 Two random placement of PUF cells on FPGA. 31
4.5 The internal structure of LUTs. The signal propagation path inside the LUTs change as the inputs change. 33
4.6 The differential signature extraction system. 38
4.7 The timing error probability for two sample PUF cells and the resulting XOR output probability under (a) normal operating condition and (b) low operating temperature of -10^oC. 39
4.8 The probability of detecting timing errors versus the input clock pulse width T. The solid line shows the Gaussian fit to the measurement data. 43

4.9 Optional caption for list of figures 44
4.10 (a) Distribution of delay parameters d_i. (b) The distribution of d_1 for normal, low operating temperature, and low core voltage. 47
4.11 The inter-chip and intra-chip response distances for $T = 0.95\ ns$ and $N_c = 2$ before (top) and after (bottom) calibration against changes in temperature. 48
4.12 The distribution of the intra- and inter-chip signature L_1 distances . . 50
4.13 Arbiter-based PUF with path swapping switches. 50
4.14 (a),(b) path swapping switch and its delay abstraction (c),(d) PDL-based switch and its delay abstraction. 53
4.15 The new arbiter-based PUF structure. 54
4.16 Reducing the response instability due to arbiter metastability by using majority voting. 55
4.17 Signal propagation delay as a function of temperature. 56
4.18 The distribution of Δ_d and stability of responses in the corresponding partitions. 57
4.19 The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL. 59
4.20 The measured delay of 32×32 circuit under tests containing a PDL with PDL control inputs being set to (a) $A_{2-6} = 00000$ and (b) $A_{2-6} = 11111$ respectively. The difference between the delays in these two cases is shown in (c). 67
4.21 Routing and placement of the PUF (a) first segment (b) last segment. 67
4.22 Measurement system setup diagram. 68
4.23 Lab setup. 68
4.24 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 6. 69

4.25 Distribution of the tuning levels across all PUF rows on all FPGAs for different operating conditions. 70
4.26 The probability of majority voting system output being equal to 1 as a function of the delay difference. 70
4.27 The sharpness (σ) of the transition slope versus the number of repetitions for majority voting. 71
4.28 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 6. . . . 71
4.29 Boxplot showing the distribution of error rates for a given operating condition corner and challenge partition. 72
4.30 Entropy of the response to the challenges at each robustness partition. 73
4.31 The correlation between effect of temperature and power supply variations on responses for 18 different scenarios. Each box plot is made of response correlation values across 12x16 PUFs. 73

5.1 The conceptual block diagram of the proposed PUF structure. 75
5.2 The proposed current based PUF system. 76
5.3 The sense amplifier output response waveform to a set of random challenges. 78
5.4 The distribution of number of '1's in responses to 100 challenges over 100 PUFs obtained from pre-layout monte-carlo simulation 79
5.5 The distribution of number of '1's in responses to 100 challenges over 100 PUFs obtained from post-layout monte-carlo simulation 80
5.6 The average response error rate as a function of the current generator transistor gate voltage obtained from pre-layout monte-carlo simulation. 82
5.7 The average response error rate as a function of the current generator transistor gate voltage obtained from pre-layout monte-carlo simulation. 83

5.8 The inter-die and intra-die response distance distribution under different usage scenarios. 84
5.9 The floor planning of the PUF components on the die. 84
5.10 (a) The PUF chip layout. (b) taped-out chip micrograph 85

6.1 (a) FPGA registration (b) Classic authentication flow (c) Time-bound authentication flow. 88
6.2 Two independent linear arbiter PUFs are XOR-mixed in order to implement an arbiter PUF with better statistical properties. 92
6.3 The 7 steps of the SlenderPUF lightweight protocol. 94
6.4 Top: random selection of an index; Middle: extracting a substring of a predefined length; Bottom: the verifier matches the received substrings to its estimated PUF response stream. 95
6.5 The probability of the arbiter PUF output flipping versus the Hamming distance between two challenge sequences for 2, 4, and 8 independent XOR-mixed PUFs [2] 103
6.6 The modeling error rate for arbiter-based PUF, and XOR PUFs with 2 and 3 outputs as a function of number of train/test CRPs. 110
6.7 True random number generation architecture based on flipflop metastability 114
6.8 Resource usage on prover and verifier sides 115

A.1 The TRNG system model. 119
A.2 The TRNG system implementation with a PI controller on FPGA. . 121
A.3 Decoding operation. 122
A.4 The complete TRNG system. 124

A.5 (a) Flip-flop operation under four sampling scenarios, (b) probability of output being equal to '1' as a function of the input signals delay difference (Δ). The numbers on the probability plot correspond to each signal arrival scenario. 125
A.6 The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL. 127
A.7 Coarse and fine PDLs implemented by a single 6-input LUT. 128
A.8 The probability of flip-flip generating a '1' output as a function of the fine and coarse tuning levels. 129
A.9 The transient counter value (decimal) versus the clock cycles. 130
A.10 Distribution of the steady state counter values and associated bit probabilities. 131

B.1 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 6. 136
B.2 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 7. 137
B.3 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 8. 138
B.4 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 9. 139
B.5 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 10. 140
B.6 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 11. 141
B.7 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 12. 142

B.8 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 13. 143
B.9 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 14. 144
B.10 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 15. 145
B.11 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 16. 146
B.12 Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 17. 147
B.13 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 6. . . . 148
B.14 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 7. . . . 149
B.15 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 8. . . . 150
B.16 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 9. . . . 151
B.17 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 10. . . . 152
B.18 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 11. . . . 153
B.19 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 12. . . . 154
B.20 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 13. . . . 155
B.21 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 14. . . . 156

B.22 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 15. . . . 157
B.23 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 16. . . . 158
B.24 Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 17. . . . 159

Tables

4.1 (a) probability of false alarm (b) probability of detection. 45
4.2 The probability of detection and false alarm before and after performing calibration on the challenge pulse width in presence of variations in temperature and core voltage. 49
4.3 18 correlation cases studies for various increments/decrements on temperature and power supply 66

6.1 List of design parameters 97
6.2 Average bit error rate of PUF in different voltage and temperature conditions in comparison with the ideal PUF output at nominal condition. 106
6.3 Average bit error rate of the Verifiers PUF model against the PUF outputs in different voltage and temperate conditions. 106
6.4 2-input XOR. 108
6.5 3-input XOR. 109
6.6 4-input XOR. 109
6.7 False rejection and acceptance error probabilities for different protocol parameters. 111
6.8 Implementation overhead on Virtex 5 FPGA 115
6.9 SHA-2 implementation overhead as reported in [3] 116

A.1 NIST Statistical Test Suite results. 133

Chapter 1

Introduction

While classic cryptography has tremendously advanced and matured over the past few decades, there are still many application domains where the computational overhead or the assumptions of the classic cryptography are limiting factors.

Classic cryptography often relies on algorithmically secure operations such as discrete logarithm and factorization and a secret key to establish security. Once these algorithms are implemented on actual physical hardware, a whole new array of vulnerabilities arise. Side channel attacks attempt to derive secret keys by non-invasively and invasively monitoring the side channel information leaked from the target system. Such attacks on the physical hardware may include monitoring and analyzing side channel information in event timing, power consumption, electromagnetic emanations, as well as fault injection attacks and direct probing. Moreover, permanent storage of secret keys requires integration of non-volatile memory such as using ROM or Flash technology. Invasive probing and scanning electron imaging can be performed to read the internal memory values that store the secret key. In addition, the computational complexity of classic cryptographic algorithms imposes a burden on applications with ultra low power requirements and resource constrained systems.

Physical unclonable functions (PUFs) provide a complimentary security paradigm for a number of application spaces where classic cryptography has shown to be inefficient or inadequate. PUFs exploit the information inherently embedded in the physical variation of the silicon devices to enable a unique chip-dependent mapping

from a set of digital inputs (challenges) to digital outputs (responses). PUFs can be employed to provide security at multiple levels and to address a range of problems from securing processors [4], to IP protection [5], and IC authentication [6].

Prior to this work, PUF implementations suffered from inefficient designs that made them unsuitable for low power applications and vulnerable to broad range of security attacks. In addition, the exiting authentication protocols because of the use hash operation and/or error correction are not suited for low power applications. This doctoral thesis presents novel formal requirements, properties, and protocols for PUFs that are used to design new architectures and implementations. The thesis starts from the foundation to formally define and derive a set of requirements and properties for physically unclonable functions. These definitions give us a tool to test, analyze and evaluate PUFs. After consolidating the definition and properties, I move from abstract concepts to concrete architectural requirements and guidelines to construct secure and efficient PUFs.

Once the requirements and desired properties are determined, robust and efficient PUFs architectures are introduced and implemented across various platforms including digital and analog ICs that conform to the introduced requirements. In particular, two PUF structures are implemented on FPGAs that leverage delay variation of digital components. The first method uses an at-speed characterization mechanism to measure component delays. The second is the long-sought implementation of arbiter-based PUF. Many efforts made by the research community to implement the arbiter-based PUF on FPGA have been previously unsuccessful, mainly because of the difficulty to achieve a symmetric routing of the arbiter PUF. The difficulty arises from the lack of freedom in routing on FPGA dictated by the rigid fabric of FPGA interconnects. In this thesis, I show the first implementation of arbiter-based PUF

on FPGA realized through a novel delay tuning mechanism of pico-second resolution.

On ASIC, an ultra-low power analog PUF implementation is presented that exploits variations in sub-threshold leakage currents of MOS devices. This is the most power efficient and smallest PUF implementation known to date. The circuit was taped out in IBM 90nm low power technology. The results show that the leakage-based PUF circuit consumes 40 femto joules to generate one bit of response. Full performance analysis and comparison are carried out on these implementations. Statistical properties and performance metrics such as response error rate in presence of temperature and voltage supply variations as well as speed, area, and power consumption are measured and reported.

Finally, I present the design of low overhead and secure protocols using PUFs. The goal of these protocols is to protect the PUF against machine learning attacks and prevent eavesdroppers or dishonest provers to pass the authentication without having access to the physical medium (PUF). Also, the protocols prevent an attacker disguised as a verifier to extract information from the PUF. The protocols are designed with elegant simplicity to dramatically lower overhead by refraining from computationally expensive classic cryptographic operations and error correction techniques. Two specific protocols, one exploiting a time bound on the authentication process and the other one utilizing a pattern matching index-based authentication on PUF responses are introduced to integrate the PUF in lightweight applications. The pattern matching protocol use a true random number generator (TRNG) to generate a nonce (number-used-once) and a random secret index. A TRNG based on flip flop metastability and a closed loop feedback system is further developed and implemented on FPGA [7].

To the best of our knowledge, this is the first comprehensive work that intro-

duces and integrates these pieces together. The contributions range from introduction of structural requirements and metrics to classify and evaluate PUFs, design of novel architectures to fulfill these requirements, implementation and evaluation of the proposed architectures on FPGAs and ASICs, and eventually integrating them into real-world security protocols.

1.1 Focus

Research work for the thesis is divided into four sub-problems.

- First, A formal conceptual and structural definition as well as a set of expected properties for PUFs are derived. Lack of consistent and structural approach to PUF design as well as disparate attempts to define desired properties and performance metrics of PUFs in the research community was the motivation behind this phase of the research work. Part of this phase was accomplished during my Master's research work [8]. Statistical and performance metrics were derived and tested on delay-based PUF in [2]. In [9], PUF implementation challenges and obstacles on FPGAs were identified. Formal definitions on PUF were developed later in [10], and [11].

- The second of part of the thesis involves implementation and evaluation of the proposed concepts and architectures on FPGA. Two candidate structures were implemented on FPGA. Prior to this research work, the only reported PUF implementation was based on ring oscillators. Ring oscillators, in addition to inefficiencies because of high power consumption and area overhead, do not satisfy the Strong PUF requirements. The first proposed structure uses at-speed test circuit and a clock sweep to measure the component delay and then maps

the delays into digital responses [12, 13, 14]. The results obtained through this research also motivated researchers in other fields to develop statistical modeling and sparse sampling tools to enable fast delay characterization and process variation modeling [15, 14].

The second implementation constructs and utilizes finely programmable delay lines to implement the arbiter PUF on FPGA [16]. Prior to this work, there have been many unsuccessful attempts to implement arbiter-based PUFs on FPGA as also identified in early phases of the research [9]. Even researchers had argued that implementation of arbiter-based PUF on FPGA is not viable [17]. Experimental results in this work reported an average response error rate of 5% . Response error rate was significantly reduced by a challenge classification method. The method only selects robust challenges that yield larger delay differences with the knowledge of the component delays.

- Third, an analog implementation based on sensing variations in sub-threshold leakage currents of MOS transistor arrays was presented in [10]. Since the main application space for PUFs is ultra low power system, I proposed an analog ASIC implementation of the PUF to further reduce the power consumption. This work was the first attempt to build Strong PUFs on analog platforms using sub-threshold leakage currents. The PUF generated digital responses by sensing the differences between minuscule leakage currents from a an array of MOS devices biased in the sub-threshold region. The leakage-based PUF is the most power efficient and smallest known to date. The performance is evaluated under various operating conditions (temperatures and voltage supplies) and challenge configurations. The lowest response error rate of 3% was achieved

when 8 currents were combined at the common gate voltage of 0.3V. The circuit was taped out in IBM 90nm low power technology. The simulation results show that the PUF circuit consumes 40 femto joules to generate one bit of response.

- Finally, two protocols are designed to enable integration of the PUF building blocks into real world lightweight applications. The first protocol is particularly designed for reconfigurable platforms such as FPGAs [12, 13]. The protocol enforces a time bound on the prover response time from the moment the device is configured. It is assumed that the reconfigurable system component delays are measured and stored through an initial registration phase. The time bound makes it practically impossible to reverse engineering the FPGA bitstream to discover the location and configuration of the PUF. Distance bounding protocols introduced in early works [18, 19] protect systems such as smart card payments and keyless entry against relaying attacks. These protocols are built upon the same concept of timing the challenge/response process. The second protocol is more generic and uses a pattern matching mechanism on the responses [20]. The pattern reveals almost no information about the original response sequence. The concept of pattern matching for reliable key generation was proposed in [21]. The index-based protocol avoids the use of computationally heavy standard cryptographic operations and error correcting codes, and thus creating an elegantly simple yet powerful authentication protocol. The proposed protocol in this thesis requires a TRNG module to generate a random index. I designed and implemented a TRNG based on flip-flop metastability and a closed loop feedback system in [7] to generate the required true random bits. The TRNG uses the programmable delay lines introduced in [16] to force the flip-flops into a metastable state. A closed-loop feedback system monitors the random output

bits and automatically adjusts the delays to correct for any statistical deviations from metastable operating point. This was the first innovative approach to use metastability of digital circuits to generate true random numbers in hardware.

1.2 Thesis Organization

The next chapter provides a preliminary background of the physical unclonable function (PUF) concept and present implementations. In the background chapter, the structural requirements of Strong PUF as well as the desired properties are further discussed. Following Chapter 2, the related literature is reviewed in Chapter 3. In particular, Chapter 3 discusses the shortcomings and merits of previous PUF implementations and protocols, as well as their vulnerabilities to attacks. Related work on implementations of true random number generation in hardware is reviewed later at the end of Chapter 3. Chapter 4 presents a delay characterization method on FPGA and demonstrates the use of the developed mechanism to implement a delay-measuring PUF. The second half of Chapter 4 harnesses the insight from observations made earlier in the chapter to build a precise delay tuning mechanism with pico-second resolution. The approach uses the programmable delay tuning system to implement arbiter-based PUF on FPGA. Comprehensive measurements and evaluations are performed on Virtex 5 FPGA on both approaches. Chapter 5 dives into the analog implementation of Strong PUF on ASICs. In this chapter, an ultra-low power PUF is introduced and implemented on 90 nano-meter IBM low power technology. Chapter 6 presents two new low power authentication protocols using PUFs. Appendix A presents the first true random number generation in hardware that uses metastability of flip-flops. Appendix B contains a set of plots which shows the measurement data taken from the twelve FPGAs in detail. Chapter 7 concludes this work.

Chapter 2

Background

Physical Unclonable Functions (PUFs) use the inherent and embedded nano- and micro-scale randomness in silicon device physics to establish and define a secret which is physically tied to the hardware. The randomness is introduced by the imperfections and lack of precise control during the fabrication process that lead to variations in device physical dimensions, doping, and material quality. The variation in device physics transfers itself into variations in electrical properties, such as transistor drive current, threshold voltages, capacitance and inductance parasitics. Such variations are unique for each IC and device on each IC. Rather than generating a static ID, PUFs typically accepts a set of input challenges and map them to a set of output responses. The mapping is a function of the unique device-dependent characteristics. Therefore, the responses two PUFs on two different chips produce to the same set of inputs are different.

A common way to build a PUF in both ASICs and FPGAs is by measuring, comparing, and quantifying the propagation delays across the logic elements and interconnects. The variations in delays appears in forms of clock skews on clock network, jitter noise on the clock, variations in setup and hold times of flipflops, and the propagation path delays through the combinational logic. On analog platforms, there is larger degree of freedom to measure variations on voltages and currents while doing the same is not feasible on digital platforms where everything is in the binary form of zeros and ones.

PUFs can be classified based on their abstract architecture which explains how challenges map to responses. In this thesis, an architectural paradigm is introduced which delineates the mapping process. The implications of such architecture is further discussed below. The architectural paradigm is used as a cornerstone to classify the PUF as what is commonly referred in the literature as Strong PUF.

Figure shows the high-level architectural block diagram that defines the construction of Strong PUF. The diagram consists of three stages. The green blocks on the lowest end generator process sensitive electric signals such as voltages, currents, and/or delays. These blocks can be as simple as a single transistor device. The yellow block above the generator block performs the selection function. The select block based on the input challenge selects a subset of the generated signals. A critical property of the select block for a Strong PUF is the ability to select an exponential number of subsets (2^n) from the set of generated signals of size n, where the maximum number of subsets is $n!$. The combine block above the select block, receives the selected subsets and combines the values in the subsets. The combination function can be a linear addition or some non-linear function. A critical observation that must be made here is that the combination must be performed in analog domain to preserve the information content of the generated signals. Since digital combination requires quantization of the signals, the entropy and information content of the combined values will be severely reduced if combination is performed in digital domain. For instance, think of adding a large number of real values and then quantizing the result, versus quantizing the real values first and perfoming the addition in digital domain. It is clear that the quantization error dramatically decreases the information content of the real number when second approach is taken. In reality, the entropy of the former approach is limited by the analog noise during the analog addition operation.

After combination is performed, the combined values are compared by the compare block and the result is represented in binary format.

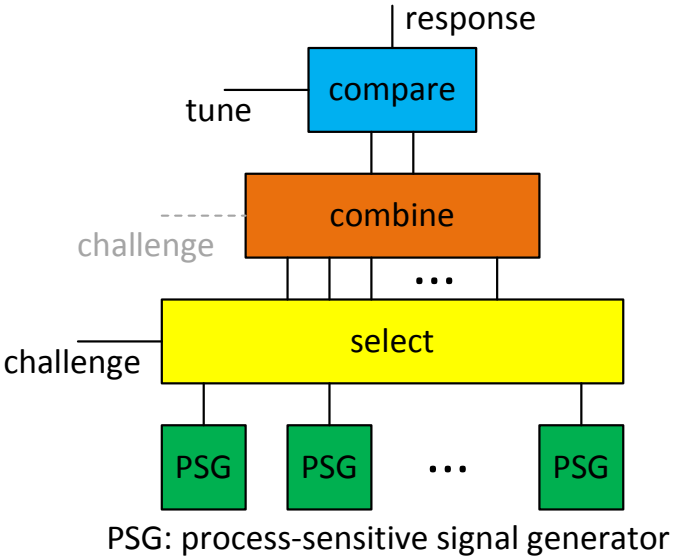


Figure 2.1 : The conceptual architecture of Strong PUF.

The work in [1] was the first to exploit the unique and unclonable delay variations of silicon devices for PUF formation. The PUF, known as arbiter PUF or delay-based PUF, is shown in Figure 2.2. The PUF uses the analog differences between the delays of two parallel paths that are identical in design and prior to fabrication, but the physical device imperfections make the delays different. The arbiter PUF follows the architectural paradigm depicted in Figure . Beginning the operations, a rising transition is exert at the PUF input producing a racing condition on the parallel paths. An arbiter at the end of the paths generates binary responses based on the signal arrival times. To enable multiple path combinations and generate an exponential number of challenge/response pairs, the paths are divided into multiple sub-paths interleaved by a set of path swapping switches. The challenges to the PUF

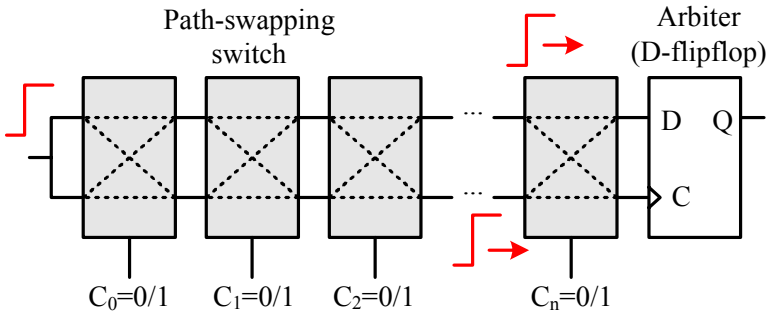


Figure 2.2 : Arbiter-based PUF introduced in [1].

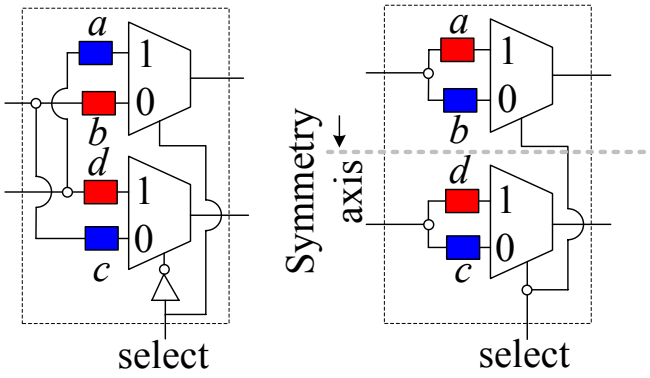
control the switches and, therefore, the varying paths are formed. Let us compare the mechanics of challenge-to-response mapping of the arbiter PUF with the diagram shown in Figure . In the arbiter PUF, the process sensitive signals are in form of delays. The selection operation is enabled by the set of path swapping switches along the signal propagation path. The combination is carried out inherently as the delays along the path add up. Therefore, the combination is simply a linear addition of the delays of the selected paths. Note that the challenges can select an exponential number of propagation paths and delays. A flip-flop at the end of the PUF compares the delay of the two paths and produces a binary result. As it can be observed, all of the components of architectural paradigm depicted in Figure can be detected in the PUF structure.

A successful implementation of this type of PUF was demonstrated on ASICs platforms [22]. It is critical to note that the differences in delays should be solely coming from manufacturing variation and not from design-induced biases. To obtain exact symmetry on the signal paths and to equalize the nominal delays, careful and precise custom layout with manual placement and routing is required for implementation on ASICs. The lack of a fine control over arbitrary placement and routing on

FPGA has resulted in difficulty in balancing the nominal delays on the racing paths within the arbiter-based PUF. Implementation on FPGA was troubled because of the constraints in routing and placement imposed by the rigid fabric of the FPGA as studied in [17, 9].

In this thesis, the problem is addressed by demonstrating a working implementation of the arbiter-based PUF on FPGA that utilizes a non-swapping symmetric switch structure as well as a precise programmable delay line (PDL) component to cancel out the systematic delay biases. The path-swapping switch previously used in the arbiter-based PUF of Figure 2.2 can be implemented by two multiplexers (MUX) and one inverter as depicted in Figure 4.14 (b). However, due to cross wiring from the lower half to the upper half (diagonal routing), maintaining symmetry in path lengths for this type of switches is extremely difficult. To avoid diagonal routing, a non-path swapping switch with a similar structure is introduced in Chapter 4 which uses two MUXes as shown in Figure 4.14 (a). As it can be seen on the figure, after applying the method the resulting routing and path lengths are symmetric and identical across the symmetry axis (drawn by the dashed line).

Another family of PUFs amenable to implementation on digital platforms and in particular FPGAs, is based on ring oscillators (RO-PUF). A ring oscillator is composed of an odd number of inverters forming a chain. Due to variations in delays of comprising logic components and interconnects, each ring oscillates at a slightly different frequency. The RO-PUF measures and compares the unique frequency of oscillation within a set of ring oscillators. A typical structure of RO-PUF is shown in Figure 2 (a). Most of the work around RO-PUFs is focused on post processing techniques, selection, quantization and comparison mechanisms to extract digital responses while achieving robustness of responses and high response entropy. It is



(a) Asymmetric path-swapping switch (b) Symmetric non path-swapping switch

Figure 2.3 : Two implementation of path selecting switches.

important to note that due to the absence of combine block in RO-PUFs and sub-exponential challenge space, RO-PUF does not pass the requirements to be classified as Strong PUF.

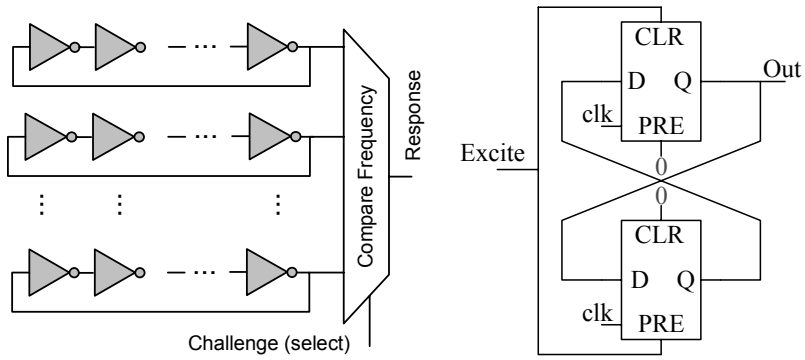
One of the early papers to consider and study ring oscillators for digital secret generation is [6]. The work proposes a 1-out-of-k mask selection scheme to enhance the reliability of generated response bits. For each k ring oscillator pairs, the pair that has the maximum frequency distance is chosen. It is argued that if the frequency difference between two ring oscillators is big enough, then it is less likely that their difference changes sign in presence of fluctuations in operating temperature or supply voltage.

In order to achieve higher stability and robustness of responses, extra information can be collected by measuring the oscillation frequency under different operating conditions. Methods presented in references [23, 24] use this information to efficiently pair or group the ring oscillators to obtain maximum response entropy. Specifically, frequency measurement is performed at two extreme (low and high) temperatures

and a linear model is built to predict the frequency at middle temperature points.

Systematic process variation can adversely affect the ability of RO-PUF for generation of unique responses. A method to improve uniqueness of ring oscillator PUF responses is discussed in [25]. A compensation method is used to mitigate the effect of systematic variation by (i) placing the group of ROs as close as possible (ii) picking the physically adjacent pair of ROs while evaluating a response bit. Large scale characterization of an array of ROs on 125 FPGAs (Spartan3E) is performed in [26]

The existing inherent race conditions in combinatorial logics with feedback loop are also used in development of other types of PUFs. For instance, a loop made of two inverter gates can have two possible states. At the power-up, the system enters into a metastable state that settles onto one of two possible states. In fact, the faster gate will dominate the slower gate and determine the output. The idea of back-to-back inverter loops is used in SRAM memory cells. SRAM-based PUFs based on the inherent race condition and variations in component delays produce unique outputs at startup. Unfortunately, in SRAM-based FPGAs, an automatic internal reset mechanism prevents using the unique startup value. A more practical implementation that is based on the same concept but uses the logic components on FPGA rather than the configuration SRAM cells, is referred to as a butterfly PUF. The basic structure of a butterfly PUF is shown in Figure 2 (b). Butterfly PUF is made of two D-flipflops with asynchronous preset and reset inputs. The flip-flops are treated as combinational logics. The work in [17] presents a comparative analysis of delay based PUF implementations on FPGA. The work particularly focuses on the requirements of maintaining symmetry in routing inside the building blocks of Arbiter-based PUF, Butterfly PUF, and RO-PUF.



(a) RO-PUF (b) Butterfly PUF

Figure 2.4 : Other delay based PUFs

Chapter 3

Related Literature

The idea of using complex unclonable features of a physical system as an underlying security mechanism was initially proposed by Pappu et al. [27]. The concept was demonstrated by studying mesoscopic physics of coherent transport through a disordered medium. Another group of researchers observed that the manufacturing process variability in modern silicon technology can be utilized for building a PUF. They proposed the arbiter-based PUF architecture based on the variations in CMOS logic delays [1].

In the arbiter-based PUF, the analog delay difference between two structurally identical parallel paths is compared. Due to manufacturing variations, the delay of these two paths are slightly different. The architecture of the arbiter-based PUF with two racing parallel paths is demonstrated in Figure 3.1. A step input simultaneously triggers the two paths. At the end of the two parallel paths, an arbiter is used to convert the analog delay difference between the paths to a digital value. The arbiter can be implemented by a D-flip flip in practice. The two paths can be divided into several smaller subpaths by inserting path swapping switches. Each set of inputs to the switches act as a challenge set (denoted by C_i), defining a new pair of racing paths whose delays can be compared by the arbiter to generate a one-bit response.

The arbiter-based PUF implementation on ASICs was demonstrated, and a number of attacks and countermeasures were discussed [1, 28, 22, 6, 29].

For implementing PUFs on FPGA, Ring oscillator (RO) PUFs were proposed [6].

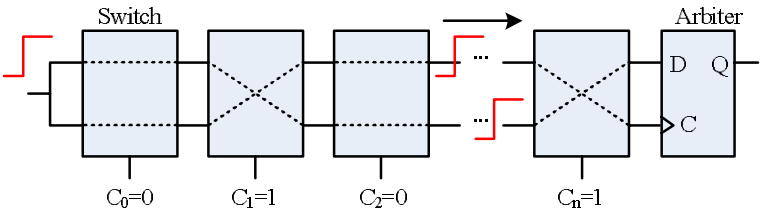


Figure 3.1 : Delay-based PUF.

Ring oscillator (RO) PUFs rely on the specific and unique delay of an oscillating path on each device [6]. The presently known PUFs of this type contain a set of ROs and a pairing mechanism to compare the their frequencies. The major drawback of the RO PUFs is having only a quadratic number of challenges with respect to the number of RO's [30] thus does not satisfy the Strong PUF requirement. Furthermore, the ROs (while in use) consume significant dynamic power due to frequent transitions during oscillations.

Another class of candidate FPGA PUFs are SRAM-PUFs and butterfly PUFs [5, 31, 29]. Each FPGA SRAM cell would naturally tend to one logic state (either zero or one) upon startup. The impact of mismatch and manufacture variability on the SRAM power-on states is utilized to extract secret digital bits. However, similar to RO-PUFs, there are only a polynomial number of challenges with respect to the number of SRAM cells. Due to the lack of analog combining mechanism and the sub-exponential size of the challenge space, SRAM PUF do not also adhere to our definition of Strong PUF. A similar work in [32] implements the same concept with nonstandard custom SRAM cells.

A digital ID extraction system based on device mismatch using an auto-zeroing comparator was introduced in [33]. The major difference between static ID generation using physical device variations and PUFs is the lack of challenges in the former to

select and combine the analog variations before quantization/digitization.

Besides the ongoing research on PUFs, several other relevant works on delay characterization serve as the enabling thrust for realization of our novel PUF structures. To perform delay characterization, Wong et al. in [34] proposed a built-in self-test mechanism for fast chip level delay characterization. The system utilizes the on-chip PLL and DCM modules for clock generation at discrete frequencies. The delay fingerprint can be used to detect any malicious modification to the original design due to insertion of hardware Trojan horses [35, 36].

In addition, the use of reconfigurability to enhance system security and IP protection has previously been a subject of research. The work in [37] proposes a secure reconfiguration controller (SeReCon) which provides secure runtime management of designs downloaded to the DPR FPGA system and protects the design IP. The work in [38] introduces methods for securing the integrity of FPGA configuration while [39] leverages the capabilities of reconfigurable hardware to provide efficient and flexible architectural support for security standards as well as defenses against hardware attacks.

3.1 Vulnerability analysis and countermeasures

PUFs have been subject to modeling attacks that breach their security and break any protocols built upon them. The basis for contemporary PUF modeling attacks is collecting a set of CRPs by an adversary, and then building a numerical or an algorithmic model from the collected data. For the attack to be successful, the models should be able to correctly predict the PUF response to any new challenge with a high probability.

In particular, it was observed that the linear arbiter-based PUF is vulnerable

to modeling attacks and the use of nonlinear feed-forward arbiters, and hashing to proposed to safeguard against this attack [1]. Moreover, error correcting codes were proposed in [40] to alleviate instability of PUF responses.

Further efforts were made to address the PUF vulnerability issues by adding input/output networks, adding nonlinearities to hinder machine learning and enforcing an upper bound on the PUF evaluation time [9, 30, 41].

Recent work on PUF modeling (reverse-engineering) used various machine learning techniques to attack both implementation and simulations of a number of different PUF families, including the realizations and simulations of linear arbiter PUFs and feed-forward arbiter PUFs [40, 42, 43, 2, 44].

The use of XORs for mixing the responses from the arbiter PUFs to safeguard them against attacks was pursued in [6]. More comprehensive analysis and description of PUF security requirements to ensure their protection against modeling attacks were presented in [45, 9]. The latest reported attacks on PUFs with k levels of XORs at their output were able to model up to $k = 5$ (after a year of running their algorithms on supercomputers) [44]. This was assuming that the full string of CRPs was known to the attacker. At the time of this publication, to the best of our knowledge, no stronger attacks on k-level XOR arbiter PUFs have been reported. We also note that, the results for $k = 5$ in [44] are for "synthetic" PUFs, not for a silicon realization of a PUF.

The use of PUF responses to create secret key for cryptographic algorithms has been explored in previous work, including [1, 40, 46, 47, 48]. Since cryptographic keys need to be stable, error correction is used for stabilizing inherently noisy PUF response bits. The classic method for stabilizing noisy PUF bits (and noisy biometrics) is error correction which is done by using helper bits or *syndrome* [49].

Since error correction needs to be robust, secure, and efficient, it is important to consider limiting the amount of secret bit leakage through the disclosed syndrome bits. A generic secure key extraction framework based on biometric data and error correction was devised in [49]. A newer information-theoretically secure Index-Based Syndrome (IBS) error correction coding for PUFs was introduced and realized in [48]. All the aforementioned methods incur a rather high overhead of error correction logic, e.g., BCH, which prohibits their usage in lightweight systems. An alternative efficient error correction method by pattern matching of responses was very recently proposed [21]. We use this pattern matching idea in our work. In [21], a 4-XOR arbiter has been used which for real PUFs has not yet been broken. Their architecture also works with a higher than 4 XOR mixing. However the error correction performance would be reduced. Their proposed protocol and application area was limited to secret key generation.

In the context of challenge-response based authentication for Strong PUFs, sending the syndrome bits for correcting the errors before hashing was investigated [1]; the necessity for error correction was due to hashing the responses before sending them to avoid reverse engineering. Naturally, the inputs to the hash have to be stable to have a predictable response. The proposed error correction methods in this context are classic error correction and fuzzy extraction techniques. Aside from sensitivity to PUF noise (because it satisfies the strict avalanche criterion) hashing has the drawback of high overhead in terms of area, delay, and power.

3.2 Hardware True Random Number Generation

The work in [50] uses sampling of phase jitter in oscillator rings to generate a sequence of random bits. The output of a group of identical ring oscillators are fed to a parity

generator function (i.e., a multi-input XOR). The output is constantly sampled by a D-flipflop driven using the system clock. In absence of noise and identical phases, the XOR output would be constant (and deterministic). However, in presence of a phase jitter, glitches with varying non-deterministic lengths appear at the output. An implementation of this method on Xilinx Virtex II FPGAs was demonstrated in [51].

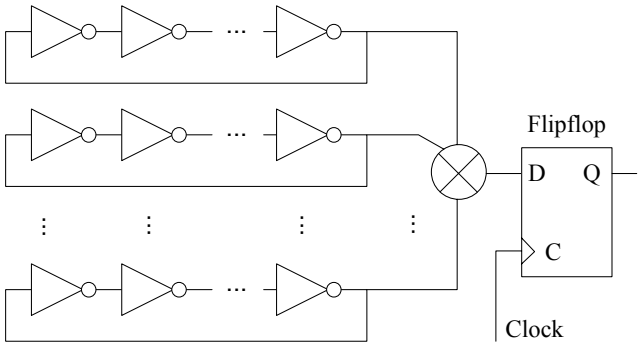


Figure 3.2 : TRNG based on sampling the ring oscillator phase jitter.

Another type of TRNG is introduced in [52] that exploits the arbiter-based Physical Unclonable Function (PUF) structure. PUF provides a mapping from a set of input challenges to a set of output responses based on unique chip-dependent manufacturing process variability. The arbiter-based PUF structure introduced in [1], compares the analog delay difference between two parallel timing paths. The paths are built identically, but the physical device imperfections make their timing different. A working implementation of the arbiter-based PUF was demonstrated on both ASICs [22] and FPGA [16, 6]. Unlike PUFs where reliable response generation is desired, the PUF-based TRNG goal is to generate unstable responses by driving the arbiter into the metastable state. This is essentially accomplished through violating the arbiter setup/hold time requirements. The PUF-based TRNG in [52] searches for

challenges that result in small delay differences at the arbiter input which then cause unreliable response bits.

To improve the quality of the output TRNG bitsteam and increase its randomness, various post-processing techniques are often performed. The work in [50] introduces resilient functions to filter out deterministic bits. The resilient function is implemented by a linear transformation through a generator matrix commonly used in linear codes. The hardware implementation of resilient function is demonstrated in [51] on Xilinx Virtex II FPGAs. The TRNG after post processing achieves a throughput of 2Mbps using 110 ring oscillators with 3 inverters in each. A post-processing may be as simple as von Neumann corrector [53] or may be more complicated such as an extractor function [54] or even a one-way hash function such as SHA-1 [55].

Besides improving the statistical properties of the output bit sequence and removing biases in probabilities, post-processing techniques increase the TRNG resilience against adversarial manipulation and variations in environmental conditions. An active adversary may attempt to bias the output bit probabilities to reduce their entropy. Post-processing techniques typically govern a trade-off between the quality (randomness) of the generated bit versus the throughput. Other online monitoring techniques may be used to assure a higher quality for the generated random bits. For instance, in [52], the generated bit probabilities are constantly monitored; as soon as a bias in the bit sequence is observed, the search for a new challenge vector producing unreliable response bits is initiated. A comprehensive review of hardware TRNGs can be found in [56]. The TRNG system proposed in this work simultaneously provides randomness, robustness, low area overhead, and high throughput.

Chapter 4

PUFs based on timing variations

4.1 Delay Signature Extraction

To measure the delays of components inside FPGA, we exploit the device reconfigurability to implement a delay signature extraction circuit. A high level view of the delay extraction circuitry is shown in Figure 4.1. The target circuit/path delay to be extracted is called the *Circuit Under Test (CUT)*. Three flip-flops (FFs) are used in this delay extraction circuit: *launch FF*, *sample FF*, and *Capture FF*. The clock signal is routed to all three FFs as shown on the Figure. Assume for now that the binary challenge input to the CUT is held constant and thus the CUT delay is fixed.

Assuming the FFs in Figure 4.1 are initialized to zero, a low-to-high signal is sent through the CUT by the launch FF at the rising edge of the clock. The output is sampled T seconds later on falling edge of the clock (T is half the clock period). Notice that the sampling register is clocked at the falling edge of the clock. If the signal arrives at the sample FF before sampling takes place, the correct signal value would be sampled; otherwise, the sampled value would be different and will generate an error. The actual signal value and the sampled value are compared by XOR logic and the result is held for one clock cycle by the capture FF.

A more careful timing analysis of the circuit reveals the relationship between the delay of the CUT (t_{CUT}), the clock pulse width (T), the clock-to-Q delay at the launch FF (t_{clk2Q}), and the clock skew between the launch and sample FFs (t_{skew}). The

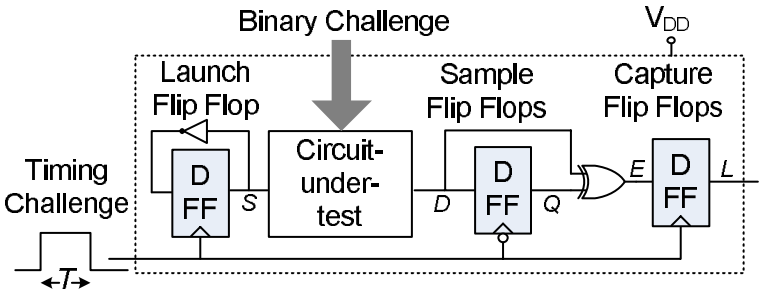


Figure 4.1 : The timing signature extraction circuit.

setup/hold of the sampling register and the setup/hold time of the capture register are denoted by t_{setS}, t_{holdS}, t_{setC}, and t_{holdC} respectively. The propagation delay of the XOR gate is denoted by t_{XOR}. The time it takes for the signal to propagate through CUT and reach the sample flip flop from the moment the launch flip flop is clocked is represented by t_P. Based on the circuit in Figure 4.1, $t_P = t_{CUT} + t_{clk2Q} - t_{skew}$.

As T approaches t_P, the sample flip flop enters a metastable operation (because of the setup and hold time violations) and its output becomes non-deterministic. The probability that the metastable state resolves to a 0 or 1 is a function of how close T is to t_P. For instance, if T and t_{CUT} are equal, the signal and the clock simultaneously arrive at the sample flip flop and the metastable state resolves to a 1 with a probability of 0.5. If there are no timing errors in the circuit, the following relationships must hold:

$$t_{holdC} < t_P < T - t_{setS} \tag{4.1}$$

The errors start to appear if t_p enters the following interval:

$$T - t_{setS} < t_P < T + t_{holdS} \tag{4.2}$$

The rate (probability) of observing timing error increases as t_p gets closer to the upper

limit of Inequality 4.2. If the following condition holds, then timing error happens every clock cycle:

$$T + t_{holdS} < t_P < 2T - (t_{setC} + t_{XOR}) \tag{4.3}$$

Observability of timing errors follows a periodic behavior. In other words, if t_p goes beyond $2T - (t_{setC} + t_{XOR})$ in Inequality 4.3, the rate of timing errors begins to decrease again. This time the decrease in the error rate is not due to the proper operation but it is because the timing errors cannot be observed and captured by the capture FF.

Inequality 4.4 corresponds to the transition from the case where timing error happens every clock cycle (Inequality 4.3) to the case where no errors can be detected (Inequality 4.5).

$$2T - (t_{setC} + t_{XOR}) < t_P < 2T + (t_{holdC} - t_{XOR}) \tag{4.4}$$

$$2T + (t_{holdC} - t_{XOR}) < t_P < 3T - t_{setS} \tag{4.5}$$

Timing errors no longer stay undetected if t_p is greater than $3T - t_{setS}$. Timing errors begin to appear and can be captured if t_p falls into the following intervals:

$$3T - t_{setupS} < t_p < 3T + t_{holdS} \tag{4.6}$$

If the following condition holds, then timing error gets detected every clock cycle.

$$3T + t_{holdS} < t_p < 4T - (t_{setC} + t_{XOR}) \tag{4.7}$$

This periodic behavior continues the same way for integer multiples of T, however it is upper bounded by the maximum clock frequency of the FPGA device. In general, if T is much larger than the XOR and flip flop delays, the intervals can be simplified to $n \times T < t_p < (n+1) \times T$ and timing errors can only be detected for odd values of n.

Notice that in the circuit in Figure 4.1, high-to-low and low-to-high transitions travel through the CUT every other clock cycle. The propagation delay of these two transitions differ in practice. Suppose that the low-to-high transition propagation delay ($t_p^{l\to h}$) is smaller than the high-to-low transition propagation delay ($t_p^{h\to l}$). Then, for low-to-high transitions, $t_p^{l\to h}$ satisfies Inequalities 4.1 and for high-to-low transitions, $t_p^{h\to l}$ satisfies Inequality 4.3. Timing errors in this case happen only for high-to-low transitions and as a result timing error can only be observed 50% of the times. Thus, the final measurement represents the superposition of both effects.

The top plot in Figure 4.2 shows the observed/measured probability of timing error as a function of clock pulse width (T). The right most region (R_1) corresponds to the error free region of operation expressed by Inequality 4.1. Note that the difference between $t_p^{h\to l}$ and $t_p^{l\to h}$ causes the plateau at R_2. The gray regions marked by R_2 and R_4 correspond to the condition expressed by Inequality 4.2. Region R_5 can be explained by Inequality 4.3. Metastable regions of R_6 and R_8 relate to inequality 4.4. Inequality 4.5 corresponds to the error free region of R_9. Similar to R_3, regions R_7 and R_{11} are due to the difference between high-to-low and low-to-high transition delays. Metastable regions of R_{10} and R_{12} relate to inequality 4.6 and lastly region R_{13} corresponds Inequality 4.7.

Notice that similar to t_p, all of the delays defined above for the XOR, flip flops, and clock skew have two distinct values for high-to-low (rising edge) and low-to-high (falling edge) transitions. Nevertheless, all of the inequalities defined in this section hold true for both cases.

We refer to the characterization circuit that includes the CUT as a *characterization cell* or simply a *cell*. Each cell in our implementation is contained in one configurable logic block (CLB). The circuit under test consists of four cascaded look-up tables

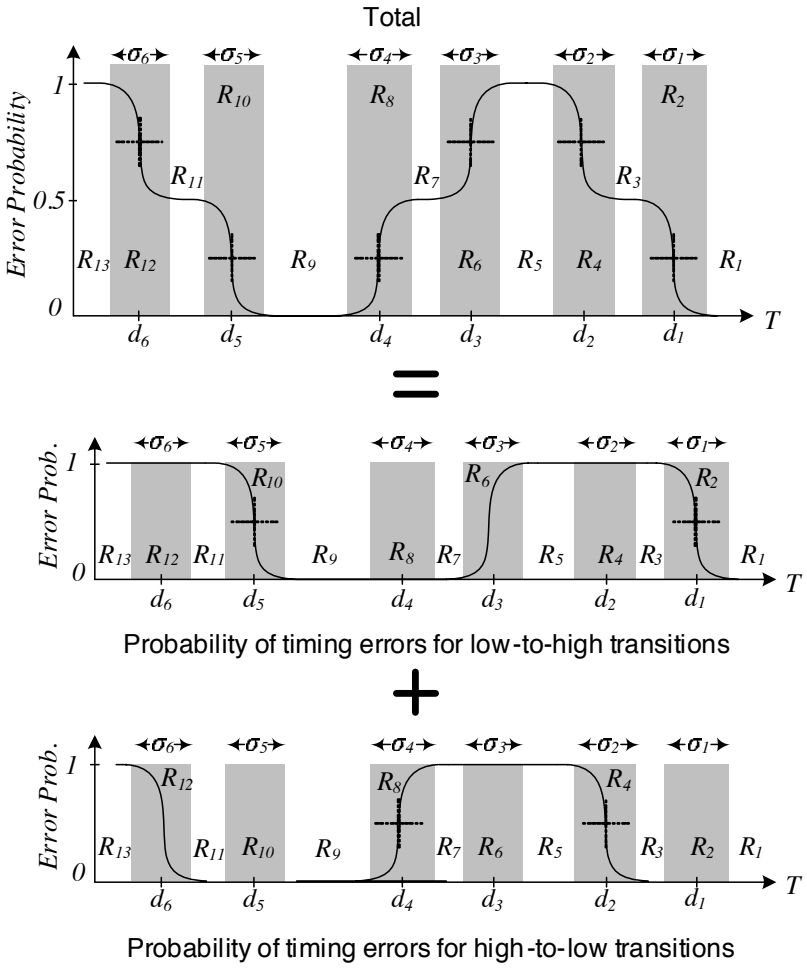


Figure 4.2 : The probability of observing timing failure as a function of clock pulse width, T.

(LUT) each implementing a variable delay inverter. We explain in Section 4.2 how the delay of the inverters can be changed.

4.1.1 Signature extraction system

In this subsection, we describe the system that efficiently extracts the probability of observing timing failure as a function of clock pulse width for a group of components on FPGA. The circuit shown in Figure 4.1 only produces a single bit flag of whether

errors happen or not. We require a mechanism to measure the rate or probability at which errors appear at the output of the circuit in Figure 4.1 to extract the smooth transitions as depicted in Figure 4.2.

To measure the probability of observing error at a given clock frequency, an error histogram accumulator is implemented by using two counters. The first counter is the error counter whose value increments by unity every time an error takes place. The second counter counts the clock cycles and resets (clears) the error counter every 2^N clock cycles, where N is the size of the binary counters. The value of the error counter is stored in the memory exactly one clock cycle before it is cleared. Now, the stored number of errors normalized to N would yield the error probability value.

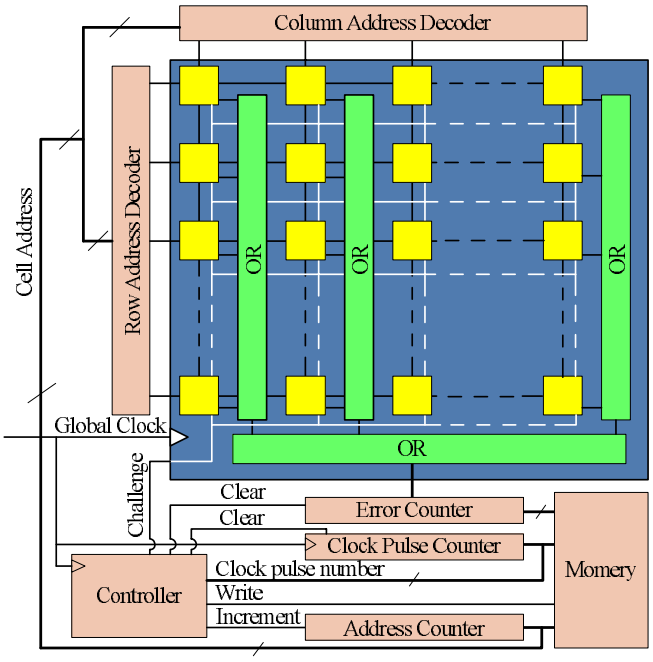


Figure 4.3 : The architecture for chip level delay extraction of logic components.

The clock frequency to the system is swept linearly and continuously in T_{sweep} seconds from $f_i = \frac{1}{2T_i}$ to $f_t = \frac{1}{2T_t}$, where $T_t < t_p < T_i$. A separate counter counts

the number of clock pulses in each frequency sweep. This counter acts as an accurate timer that bookmarks the frequency at which timing errors happen. The value of this counter is retrieved every time the error counter content is written into memory. This action happens every 2^N clock cycles. For further details on clock synthesis see [14].

The system shown in Figure 4.3 is used for extracting the delays of an array of CUTs on the FPGA. Each square in the array represents the characterization circuit (or *cell*) shown in Figure 4.1. Any logic configuration can be utilized within the CUT in the characterization circuit. In particular, the logic inside the CUT can be made a function of binary challenges, such that its delay varies by the given inputs. The system in Figure 4.3 characterizes each cell by sweeping the clock frequency once. Then, it increments the cell address and moves to the next cell. The cells are characterized in serial. The row and column decoders activate the given cell while the rest of the cells are deactivated. Therefore, the output of the deactivated cells remain zero and the output of the OR function solely reflect the timing errors captured in the activated cell. Each time the data is written to the memory, three values are stored: the cell address, the accumulated error value, and the clock pulse number at which the error has occurred. The clock counter is then for each new sweep. The whole operation iterates over different binary challenges to the cells. Note that the scanning can also be performed in parallel to reduce the characterization time [14].

4.1.2 Characterization accuracy

The timing resolution, i.e., the accuracy of the measured delays, is a function of the following factors: (i) the clock jitter and noise, (ii) the number of frequency sample points, and (iii) the number of pulse samples at each frequency. Recall that

the output of the characterization circuit is a binary zero/one value. By resending multiple clock pulses of the same width to the circuit and summing up the number of ones at the output, a real-valued output can be obtained. The obtained value represents the rate (or the probability when normalized) at which the timing errors happen for the input clock pulse width. Equivalently, it represents a sample point on the curve shown in Figure 3. The more we repeat the input clock pulse, the higher sample resolution/accuracy can be achieved along Y axis. Now suppose that the clock pulse of width T is sent to the PUF for M times. Due to clock jitter and phase noise, the characterization circuit receives a clock pulse of width $T_{eff} = T + T_j$, where T_j is additive jitter noise. Let us assume T_j is a random variable with zero mean and a symmetric distribution. Since the output probability is a smooth and continuous function of T_{eff}, estimating the probability by averaging will be an asymptotically unbiased estimator as $M \to \infty$. Finally, the minimum measurable delay is a function of the maximum speed at which the FFs can be driven (maximum clock frequency). When performing a linear frequency sweep, a longer sweep increases (ii) and (iii) and thus the accuracy of the characterization. A complete discussion on characterization time and accuracy for this method is presented in [14].

4.1.3 Parameter extraction

So far, we have described the system that measures the probability of observing timing errors for different clock pulse widths. The error probability can be represented compactly by a set of few parameters. These parameters are directly related to the circuit component delays and flip flop setup and hold time. It can be shown that the probability of timing error can be expressed as the sum of shifted Gaussian CDFs [9]. The Gaussian nature of the error probabilities can be explained by the central limit

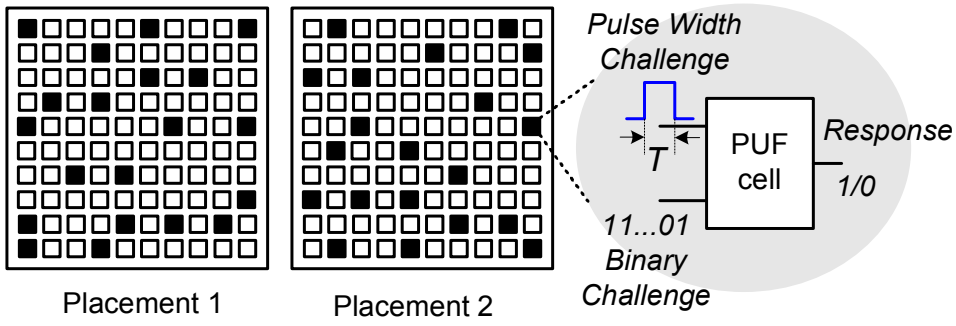


Figure 4.4 : Two random placement of PUF cells on FPGA.

theorem. Equation 4.8 shows the parameterized error probability function.

$$f_{\mathbf{D},\Sigma}(t) = 1 + 0.5 \sum_{i=1}^{|\Sigma|-1} -1^{\lceil i/2 \rceil} \left[Q(\frac{t - d_i}{\sigma_i}) \right] \tag{4.8}$$

where $Q(x)=\frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right)$ and $d_{i+1} > d_i$. To estimate the timing parameters, f is fit to the set of measured data points (t_i,e_i), where e_i is the error value recorded when the pulse width equals t_i.

4.2 Timing PUF

To enable authentication, a mechanism for applying challenge inputs to the device and observing the evoked responses is required. In this section, we present a PUF circuit based on the delay characterization circuit shown in Figure 4.1. The response is a function of the clock pulse width T, the delay of circuit under test, t_{CUT}, and flip flop characteristics, σ_i. In the following, we discuss three different ways to challenge the PUF.

4.2.1 Pulse challenge

One way to challenge the PUF is to change the clock pulse width. The clock pulse width can be considered as an analog input challenge to the circuit in Figure 4.1. The response to a given clock pulse of width T is either 0 or 1 with the probability given by Equation 4.8 or the plot in Figure 4.2.

However, the use of clock pulse width as challenge has a number of implications. First, the response from the PUF will be predictable if T is either too high and too low compared to the nominal circuit under test delay t_{CUT}. Predictability of responses makes it easy for the attacker to impersonate the PUF without knowledge of the exact value of t_{CUT}. As another example, suppose that the response to multiple clock pulses of the same width, T_1, are equal to '0'; then, the attacker can deduce that T_1 is in either region R_1 or R_9 in Figure 4.2 with high confidence. If the nominal boundaries of these regions (R_1,...,R_{13}) are known, the attacker can determine which region T_1 belongs by just comparing it to the boundaries $T_{R_i} < T_1 < T_{R_{i+1}}$. Knowing the correct region, it becomes much easier to predict the response to the given pulse width, especially for odd regions $R_1, R_3, ..., R_{13}$.

Within the thirteen regions shown in Figure 4.2, the six regions that include transitions produce the least predictable responses. Setting the challenge clock pulse width to the statistical median of the center points of transitions in Figure 4.2 would maximize the entropy of the PUF output responses. In other words, there are only six independent pulse widths that can be used for challenges and the results for other pulse widths are highly predictable. As it can be seen, the space of possible independent challenges for this type of challenge is relatively small.

Another limitation of pulse challenges is that depending on the available clocking resources, generating many clock pulses with specific widths can be costly. Under

such limitations, the verifier may prefer to stick to a fixed pulse width. In the next sections, we look into other alternatives to challenge the PUF.

4.2.2 Binary challenge

An alternative method to challenge the PUF is to change the t_{CUT} while the clock pulse width is fixed. So far, we assumed that the delay of CUT is not changing. To change t_{CUT}, one must devise an input vector to the circuit-under-test that changes its effective input/output delay by altering the signal propagation path inside the CUT. In other words, the binary input challenge vector alters the CUT delay by changing its internal signal propagation path length, hence affecting the response.

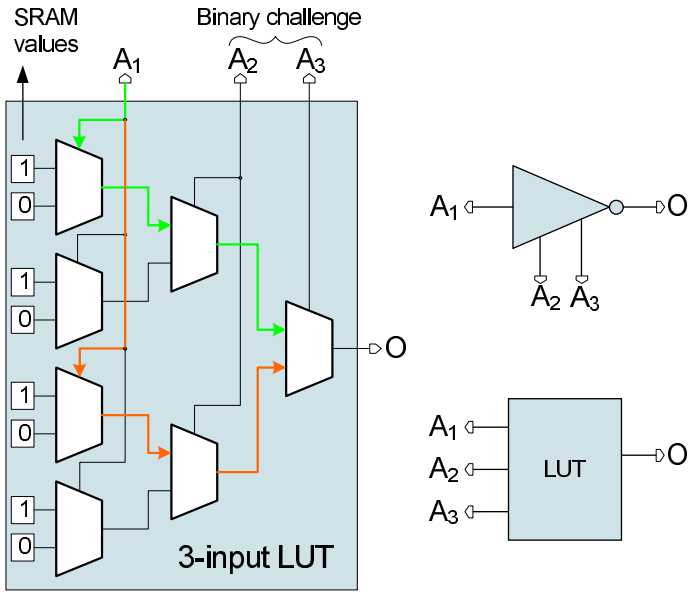


Figure 4.5 : The internal structure of LUTs. The signal propagation path inside the LUTs change as the inputs change.

In this work, we introduce a low overhead method to alter the CUT delay by tweaking the LUT internal signal proportion path. We implement the CUT by a

set of LUTs each implementing an inverter function. Figure 4.5 shows the internal circuit structure of an example 3-input LUT. In general, a Q-input LUT consists of 2^Q-1 2-input MUXs which allow selection of 2^Q values stored in SRAM cells. The SRAM cell values are configured to implement a pre-specified functionality.

In this example, the SRAM cell values are configured to implement an inverter. The LUT output is only the function of A_1, i.e., $O = f(A_1)$, disregarding values on A_2 and A_3. However, changing the inputs A_2 and A_3 can alter the delay of the inverter due to the modifications in the signal propagation paths inside the LUT. For instance, two internal propagation path for the values of $A_2A_3 = 00$ and $A_2A_3 = 11$ are highlighted in Figure 4.5. As it can be seen, the path length for the latter case is longer than the former, yielding a larger effective delay. The LUTs in Xilinx Virtex 5 FPGAs consist of 6 inputs. Five inputs of the LUT can be used to control and alter the inverter delay resulting in $2^5 = 32$ distinct delays for each LUTs. Finally, note that the delays for each binary input must be measured prior to authentication. The response to the PUF is then predicted by the verifier based on the configured delay and the input clock pulse width.

4.2.3 Placement challenge

Another important type of challenge which can be implemented solely on reconfigurable platforms is the placement challenge. This type of challenge is enabled by the degree of freedom in placing the PUF cells on FPGA in each configuration. During characterization, a complete database of all CUT delays across the FPGA is gathered. At the time of authentication, only a subset of these possible locations within the FPGA array are selected to implement and hold the PUF cells. The placement challenge is equivalent to choosing and querying a subset of PUF cells, where the

selection input is embedded in the configuration bitstream.

Figure 4.4 shows two random placements of 20 PUF cells across the FPGA array. Each black square in the figure contains a PUF cell which receives a pulse and binary challenge. The high degree of freedom in placement of PUF cells across the FPGA results in a huge challenge/response space. In our implementation, each PUF cell can be fit into a CLB on FPGA. With N CLBs on FPGA, there will be $\binom{N}{k}$ different ways to place k PUF cells on FPGA. The smallest Xilinx Virtex 5 FPGA (LX30) has 2400 CLBs which enables $\binom{2400}{512}$ number of possibilities to place 512 PUF cells on the FPGA.

4.3 Response robustness

Although PUF responses are functions of chip-dependent process variations and input challenges, they can also be affected by variations in operational conditions such as temperature and supply voltage. In this section, we discuss two techniques to provide calibration and compensation to make responses resilient against variations in operational conditions.

The first method takes advantage of on-chip sensors to perform linear calibration of the input clock pulse width challenge, while the second method uses a differential structure to cancel out the fluctuations in operational conditions and extract signatures that are less sensitive to variations in operational conditions. We will discuss the advantages and disadvantages of each method. The existing body of research typically addresses this issue mainly through the use of error correction techniques [40] and fuzzy extractors [46]. The error correction techniques used for this purpose rely on a syndrome which is a public piece of information being sent to the PUF system along with the challenge. The response from the PUF and the syndrome are

input to the ECC to produce the correct output response. The methods discussed in this section help reduce the amount of errors in responses and they can be used along with many other error correction techniques.

4.3.1 Linear Calibration

The extracted delay signatures at characterization phase are subject to changes due to aging of silicon devices, variations in the operating temperature, and supply voltage of the FPGA. Such variations can undermine the reliability of the authentication process. The proposed method performs calibration on clock pulse width according to the current operating conditions. Fortunately, many modern FPGAs are equipped with built-in temperature and core voltage sensors. Before authentication begins, the prover is required to send to the verifier the readings from the temperature and core voltage sensors. The prover, then based on the current operating conditions, adjusts and calibrates the clock frequency. The presented calibration method linearly adjusts the pulse width using the Equations 4.9 and 4.10.

$$T_{calib} = \alpha_{tmp} \times (tmp_{cur} - tmp_{ref}) + T_{ref} \tag{4.9}$$

$$T_{calib} = \alpha_{vdd} \times (vdd_{cur} - vdd_{ref}) + T_{ref} \tag{4.10}$$

tmp_{ref} and vdd_{ref} are the reference temperature and FPGA core voltage measured during the characterization phase. tmp_{cur} and vdd_{cur} represent the current operating conditions. The responses from the PUF to the clock pulse width T_{calib} are then treated as if T_{ref} were sent to the PUF at reference operating condition. The calibration coefficients α_{tmp} and α_{vdd} are device specific. These coefficients can be determined by testing and characterizing each single FPGA at different temperatures and supply voltages. For example, if $d_i^{tmp_1}$ and $d_i^{tmp_2}$ are i-th extracted delay parameter under

operating temperatures tmp_1 and tmp_2, then

$$\alpha_{tmp,i} = \frac{d_i^{tmp_1} - d_i^{tmp_2}}{tmp_1 - tmp_2}, \alpha_{vdd,i} = \frac{d_i^{vdd_1} - d_i^{vdd_2}}{vdd_1 - vdd_2} \tag{4.11}$$

Note that for each delay parameter on each chip, two calibration coefficients can be defined (one for temperature and one for voltage supply effect) and the clock pulse width can be calibrated accordingly. Ideally, with the help of a more sophisticated prediction model (potentially a nonlinear model) trained on a larger number of temperature and voltage supply points (instead of two points as in Equation 4.11), highly accurate calibration can be performed on the clock frequency. In reality, due to limitations on test time and resources, it is impractical to perform such tests for each FPGA device. Instead, calibration coefficients can be derived from a group of sample devices and a universal coefficient can be defined for all devices by averaging the coefficients. In Section A.2, we demonstrate reliability of authentication for universal calibration coefficients. Note that in Equations 4.9 and 4.10, we assume that only one type of operational condition variation is happening at a time and both temperature and voltage supply do not fluctuate simultaneously. However, if we consider these effects independently, we can superimpose the effects by applying Equation 4.9 to the output of Equation 4.10. A more general approach would be to consider a 2D nonlinear transformation given by:

$$T_{calib} = f(vdd_{cur}, tmp_{cur}, T_{ref}) \tag{4.12}$$

The main disadvantage of calibration methods is the time and effort required to characterize the delay at various operational conditions. Hence, more effort spent on building and training the regression model, the more accurate calibration and a higher robustness in responses can be achieved.

4.3.2 Differential Structure

In this section, we introduce a differential PUF structure, that compensates for the common mode variation induced by the impact of fluctuations in operational conditions on the delays. The goal of the method is to extract a signature that is invariant to fluctuations in operational conditions.

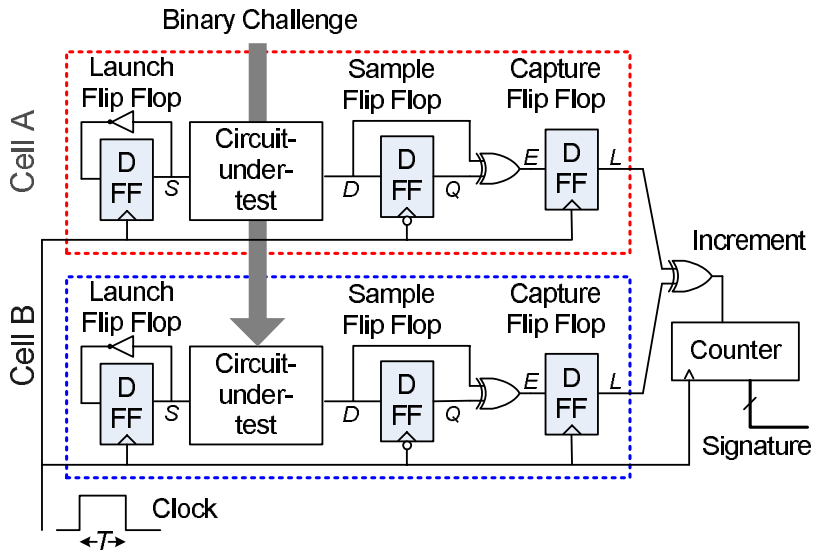


Figure 4.6 : The differential signature extraction system.

The PUF introduced previously receives a clock pulse and a binary challenge to produce a binary response. Here, instead of looking at the output responses from a single PUF cell, we consider the difference of the responses from two adjacent PUF cells. More specifically, the outputs of the capture flip flops from the two cells drive an XOR logic. Assuming i_1 and i_2 are the inputs and O is the output of the XOR logic, then the probability of output being equal to '1' , ρ_O, as a function of the probability of inputs being equal to '1', ρ_1 and ρ_2, can be written as:

$$\rho_O = \rho_1 + \rho_1 - 2 \times \rho_1 \times \rho_2 \tag{4.13}$$

ρ_1 and ρ_2 are functions of the clock pulse width (T) and the binary challenge as explained in Section 4.1. The resulting output probability is shown in Figure 4.7 (see the red dashed line) for two sample PUF cells under (a) normal operating condition and (b) low operating temperature of -10^oC. As it can be seen, since both PUF cell delay parameters are shifted together under the same operational conditions, the resulting XOR output probability retains the shape, with only a scalar shift along the x axis. To extract robust signatures, one needs to look into shift invariant features that are less sensitive to environmental variables. Features such as the high/low region widths of the resulting XOR probability plot, or the total area under the XOR output probability plot can be used for this purpose.

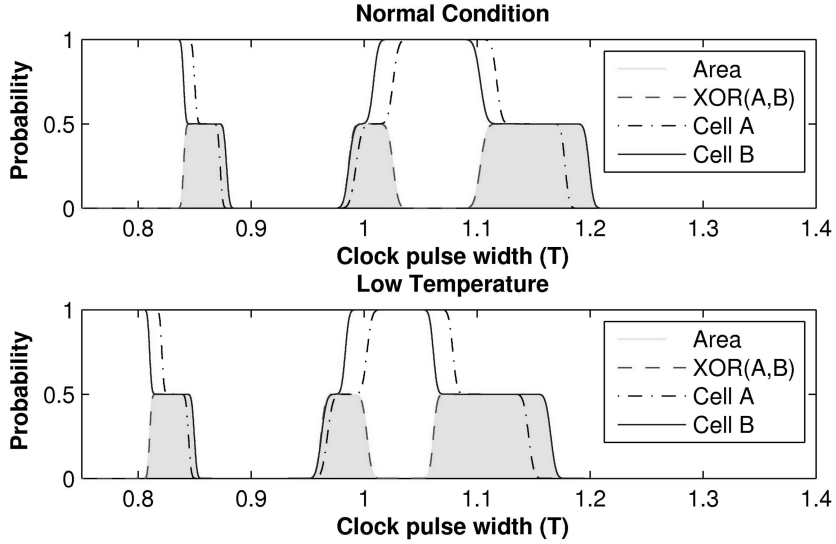


Figure 4.7 : The timing error probability for two sample PUF cells and the resulting XOR output probability under (a) normal operating condition and (b) low operating temperature of -10^oC.

In this work, we use the area under the XOR output probability curve. The area is shaded in Figure 4.7 for the two operating conditions. The area under the curve can be calculated by integrating the probability curve from the lowest to highest

clock pulse width. We use the Riemann sum method to approximate the total area underneath the XOR probability curve in hardware. The result of the integration is a resilient real valued signature extracted from the PUF cell pairs.

In order to find a quick approximation to this integral in hardware, we sweep the input clock frequency linearly from frequency $f_l = 1/2T_u$ to $f_u = 1/2T_l$ where $T_l \ll D_{min}$, $T_u \gg D_{max}$, D_{min} and D_{max} represent lowest and highest bounds on delay parameters under all operational conditions. In other words, the sweep window must always completely contain all parts of the curve. The output of the XOR is connected to a counter as shown in Figure 4.6. The aggregate counter value after a complete sweep is a function of the area under the curve. Please note that this value is not exactly equal to the area under the curve and is only proportional to the integral. Also, a longer sweep time results in a larger number of clock pulses and thus more accurate approximation of the signature. This is analogous to using a larger number of narrower subintervals when approximating the area under curve with the Riemann sum to achieve a smaller approximation error.

Although the generated responses are less sensitive to variations in operational conditions, it should be noted that the responses are a function of the difference in the timing characteristics of the two PUF cells. The area under the curve loses a lot of information about the shape of the curve and also some information is lost on each individual probability curve through the difference operation. Therefore, the responses have a lower entropy compared to the linear calibration method. To obtain the same amount of information, more PUF cell pairs must be challenged and scanned. Another limitation of this structure is the length of the input challenge. To estimate the area under the curve with a high accuracy, the whole interval from the lowest to the highest frequency must be swept in fine steps and thus, it would require

more clock pulses compared to the other method. Using few clock pulses leads to a larger area estimation error, lower probability of detection, and higher probability of false alarm. Finally, the pairing of the PUF cells introduces another degree of freedom to the system where a set of challenges can specify pairing of the PUF cells.

4.4 Experimental evaluations

In this section, the implementation details of the signature extraction system are presented. We demonstrate results obtained by measurements performed on Xilinx FPGAs and further use the platform to carry out authentication on the available population of FPGAs. For delay signature extraction, the system shown in Figure 4.3 is implemented on Xilinx Virtex 5 FPGAs. The system contains a 32× 32 array of characterization circuits as demonstrated in Figure 4.1. The CUT inside the characterization circuit consists of 4 inverters each being implemented using one 6-input LUT. The first LUT input (A_1) is used as the input of the inverter and the rest of the LUT inputs (A_2,...,A_6) serve as the binary challenges which alter the effective delay of the inverter. The characterization circuit is pushed into 2 slices (one CLB) on the FPGA. In fact, this is the lower bound on the characterization circuit hardware area. The reason is that the interconnects inside the FPGA force all the flip flops within the same slice to operate either on rising edge or falling edge of the clock. Since the launch and sample flip-flops must operate on different clock edges, they cannot be placed inside the same slice. In total, 8 LUTs and 4 flip flops are used (within two slices) to implement the characterization circuit. The error counter size (N) is set to 8. To save storage space, the accumulated error values are stored only if they are between 7 and 248.

We use an ordinary desktop function generator to sweep the clock frequency from

8MHz to 20MHz and afterwards shift the frequency up 34 times using the PLLs inside the FPGA. The sweeping time is set to 1 milli seconds (due to the limitations of the function generator, a lower sweeping time could not be reached). The measured accumulated error values are stored on an external memory and the data is transferred to a PC for further processing. Notice that the storage operation can easily be performed without the logic analyzer by using any off-chip memory.

The system is implemented on twelve Xilinx Virtex 5 XC5VLX110 chips and the measurements are taken under different input challenges and operating conditions. The characterization system in total uses 2048 slices for the characterization circuit array and 100 slices for the control circuit out of 17,280 slices.

The measured samples for each cell are processed and the twelve parameters as defined in Section 4.1.3 are extracted. Figure 4.8 shows the measured probability of timing error versus the clock pulse width for a single cell and a fixed challenge. The (red) circles represent original measured sample points and the (green) dots show the reconstructed samples. As explained earlier, to reduce the stored data size, error samples with values of 0 and 1 (after normalization) are not written to the memory and later are reconstructed from the rest of the sample points. The solid line shows the Gaussian fit on the data as expressed in Equation 4.8.

Parameter extraction procedure is repeated for all cells and challenges. Figure 4.9 shows the extracted parameters d_1 and σ_1 for all cells on chips #9 and #10 while the binary challenge is fixed. The pixels in the images correspond to the cells within the 32×32 array on FPGA. Some levels of spatial correlation among d_1 parameters can be observed on the FPGA fabric.

The boxplots in Figure 5.10(a) show the distribution of the delay parameters d_i for i=1,2,...,6 over all 12 chips and 1024 cells and 2 challenges. The central mark on

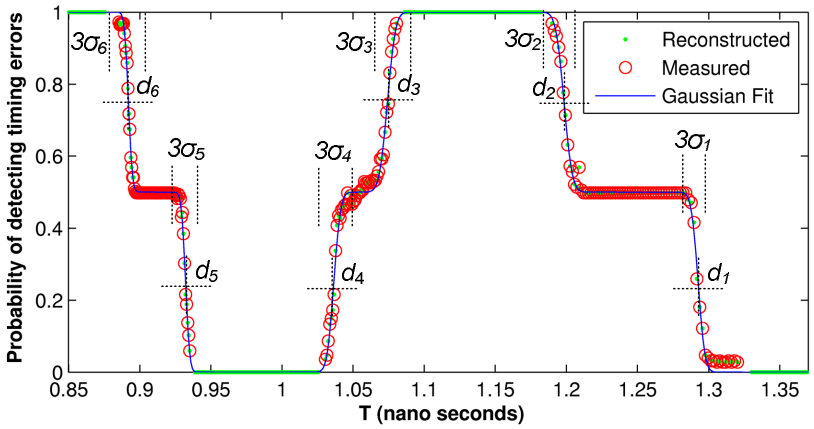


Figure 4.8 : The probability of detecting timing errors versus the input clock pulse width T. The solid line shows the Gaussian fit to the measurement data.

the boxplot denotes the median, the edges of the boxes correspond to the 25th and 75th percentiles, the whiskers extent to the most extreme data points and the plus signs show the outlier points.

Using the measured data from the twelve chips, we investigate different authentication scenarios. The authentication parameters substantially increase the degree of freedom in challenging the PUF. These parameters include the number of clock pulses to send to the PUF (N_p), the number of binary challenges to apply to the PUF (N_c), the challenge clock pulse width (T), and the number of PUF cells (N_{cell}) to be queried. In other words, in each round of authentication, N_c challenges are applied to N_{cell} PUF cells on the chip and then N_p pulses of width T are sent to to these PUF cells. The response to each challenge consists of N_p bits. For ease of demonstration, the response can be regarded as the percentage of ones in the N_p response bits, i.e., an integer between 0 and N_p.

To quantify the authentication performance, we study the effect of N_{cell} and T on the probability of detection (p_d) and false alarm (p_f). Detection error occurs in

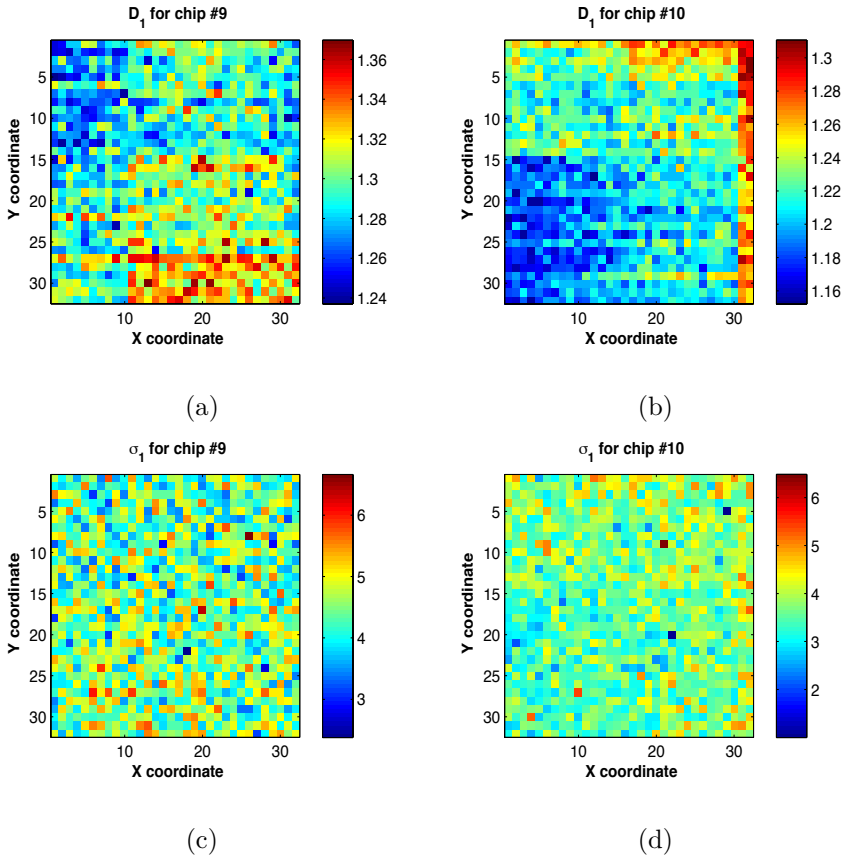


Figure 4.9 : The extracted delay parameters d_1 (a,b) and σ_1 (c,d) for chips 9 and 10.

cases where the test and target chips are the same, but due to instability and noise in responses, they fail to be authenticated as the same. On the other hand, false alarm corresponds to the cases where the test and target chips are different, but they are identified as the same chips. During this experiment, the binary challenges to PUF cells are fixed and the number clock pulses is set to $N_p = 8$. The clock width (T) is set to each of the medians of the values shown in Figure 5.10 (a). Setting the clock pulse width to the median values results in least predictability of responses. All N_{cell}=1024 PUF cells are queried. The same experiment is repeated for 10 times to obtain 10 response vectors (each vector is $N_p = 8$ bits) for each chip.

Therefore, each clock pulse generates 8×1024 bits of responses from every chip. After that, the distance between the responses from the same chips (intra-chip distance) over repeated evaluations is measured using the normalized L_1 distance metric. The distance between responses from different chips (inter-chip distance) is also measured. If the distance between the test chip and the target chip responses is smaller than a pre-specified detection threshold, then the chip is successfully authenticated. In the experiments, the detection threshold is set at 0.15.

Table 4.1 shows the probability of detection and false alarm for different clock pulse widths and number of queried PUF cells. To calculate the probabilities, the distance between the response of every distinct pair of FPGAs are calculated. The number of pairs with a response distance of less than 0.15, normalized to the total number of pairs yield the probability of false alarm. To find the probability of detection, the distance between the responses from the same chip acquired at different times are compared to 0.15. The percentage stay within the threshold determine the probability of detection. As it can be observed, the information extracted from even the smallest set of cells is sufficient to reliably authenticate the FPGA chip if the pulse width is correctly set.

(a)

N_{cell}	Challenge Pulse Width					
	1.23	1.15	1.06	1.03	0.9	0.87
64	0.96	0	0	0	0	1.52
128	2.04	0	0	0	0	1.52
256	4.55	0	0	0	0	1.52

(b)

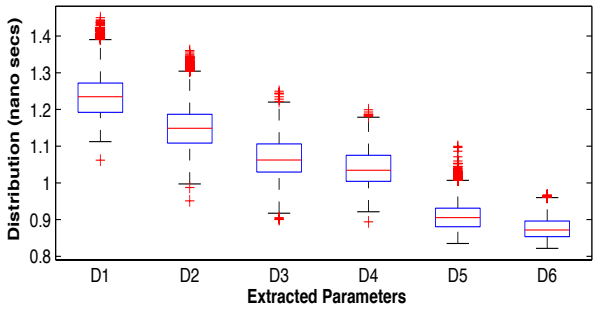
N_{cell}	Challenge Pulse Width					
	1.23	1.15	1.06	1.03	0.9	0.87
64	93.3	96.2	100	100	100	100
128	94.2	98.8	100	100	100	100
256	99.85	100	100	100	100	100

Table 4.1 : (a) probability of false alarm (b) probability of detection.

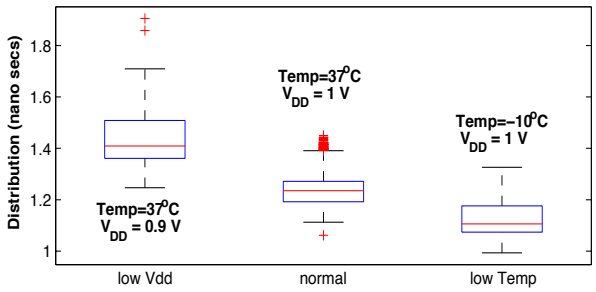
In the next experiment, we study the effect of fluctuations in the operating conditions (temperature and core supply voltage) on the probabilities of detection and false alarm. Moreover, we demonstrate how linear calibration of the challenge clock pulse width can improve the reliability of detection. To calculate the calibration coefficient defined by Equation 4.11, we repeat the delay extraction process and find the delay parameters for all twelve chips at temperature -10^oC and core voltage 0.9 Volts. The chip operates at the temperature 37^oC and core voltage of 1 volts in the normal (reference) condition. We use the built-in sensors and the Xilinx Chip Scope Pro package to monitor the operating temperature and core voltage. To cool down the FPGAs, liquid compressed air is consistently sprayed over the FPGA surface. Figure 5.10 (b) depicts the changes in the distribution of the first delay parameter (d_1) at the three different operating conditions.

The probabilities of detection and false alarm are derived before and after performing calibration on the challenge pulse width for different clock pulse widths and number of binary challenges to the cells. In this experiment, all 1024 PUF cells on the FPGA are queried for the response. N_p =8 as before. As it can be seen in Table 4.2, the detection probabilities are significantly improved after performing linear calibration based on the coefficients extracted for each chip. The variables v_{low} and t_{low} correspond to -10^oC temperature and 0.9 supply voltages respectively. The reported probabilities of Table 4.2 are all in percentage. Also note that for the challenge pulse width of $T = 0.87\ ns$, the probability of detection reaches 100% and probability of alarm falls to zero after calibration. The same holds true for $N_c = 2$ and $T = 0.87, 0.9, 0.95$. Thus, increased level of reliability can be achieved during authentication with proper choice of pulse width and number of challenges.

Figure 4.11 shows how performing calibration decreases the intra-chip response



(a)



(b)

Figure 4.10 : (a) Distribution of delay parameters d_i. (b) The distribution of d_1 for normal, low operating temperature, and low core voltage.

distances in presence of temperature changes. The histogram corresponds to $T = 0.95ns$ and $N_c = 2$ in Table 4.2 before and after calibration.

Next, we examine the differential signature extraction system presented in Section 4.3.2. To extract the signature, the base frequency is swept from 8 to 20 MHz in a linear fashion in 1 mili second and shifted up 34 times using the FPGA internal PLLs. The sweep is repeated for the 512 pairs of PUF cells producing a real-valued signature vector of size 512. A large number of pulses ($\sim 10^7$) are generated in a complete sweep. The signature as explained in Section 4.3.2 is the accumulation of the timing errors over a complete sweep. To achieve an accurate approximation

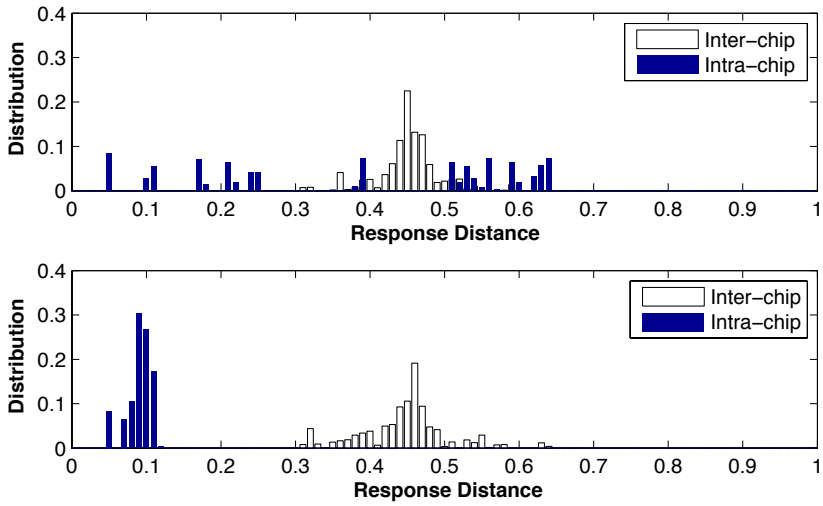


Figure 4.11 : The inter-chip and intra-chip response distances for $T = 0.95\ ns$ and $N_c = 2$ before (top) and after (bottom) calibration against changes in temperature.

of the area under the curve, a large number of clock pulses must be tried. This is the main disadvantage of this method compared to the singled ended method. To extract the shift invariant parameters such the region width and/or area under the probability curve probing the PUF circuit at single frequency points will not yield sufficient information. Therefore, a complete sweep covering the regions with high information content is needed. The L_1 distance of the signatures from the same chip under different operational conditions (intra-chip distance) and the distance of the signatures from different chips (inter-chip distance) are calculated. Figure 4.12 shows the distribution of intra and inter-chip distance of signatures under variations in temperature and supply voltage for the twelve Virtex 5 chips. As it is shown in the figure, the distance among signatures obtained at room temperature and -10^oC temperature from the same chip is always smaller than those from different chips, resulting in 100% probability of detection and 0% false alarm probability. However,

		No Calibration								Calibrated							
		N_C=1				N_C=2				N_C=1				N_C=2			
		v_{low}		t_{low}		v_{low}		t_{low}		v_{low}		t_{low}		v_{low}		t_{low}	
		p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f
T	1.23	18.4	0	33.3	16.7	18.4	0	33.3	22.29	100	0	75	0	100	0	75	0
	1.06	18.4	0	18.4	0	18.4	0	18.4	0	50	0	50	0	57.3	0	50	0
	1.01	18.4	0	16.7	0	18.4	0	16.7	0	66.6	0	75	0	68.2	0	75	0
	0.95	18.4	0	16.7	0	18.4	0	16.7	0	66.7	0	100	0	84.9	0	100	0
	0.9	16.7	0	25	0	16.7	0	25	0	83.3	0	91.7	0	83.4	0	100	0
	0.87	25	0	25	0	25	1.5	25	0	100	0	100	0	100	0	100	0

Table 4.2 : The probability of detection and false alarm before and after performing calibration on the challenge pulse width in presence of variations in temperature and core voltage.

with 10% variations in voltage supply, the intra- and inter-chip distributions overlap slightly.

4.5 Arbiter PUF on FPGA

One of the major problems in implementation of PUFs on FPGAs, particulary the arbiter-based PUFs, is in signal routing. Unlike ASICs where hand-drawn custom layout is possible, routing on FPGA is constrained by its rigid fabric and interconnect structure. As a result, performing completely symmetric routing is physically infeasible in most cases. The PUF designer may do his/her best to constrain and guide the placement and routing software to achieve the highest degree of symmetry in the PUF layout. However, due to physical constraints of the FPGA fabric, the designer may still not be able to achieve complete symmetry on some routes. Asymmetries in routing when implementing PUFs can lead to bias in delay differences leading to predictable responses, lack of randomness, and decreased response entropy [17, 9].

The PUF routing can be divided into four different sections; the routing (1) before

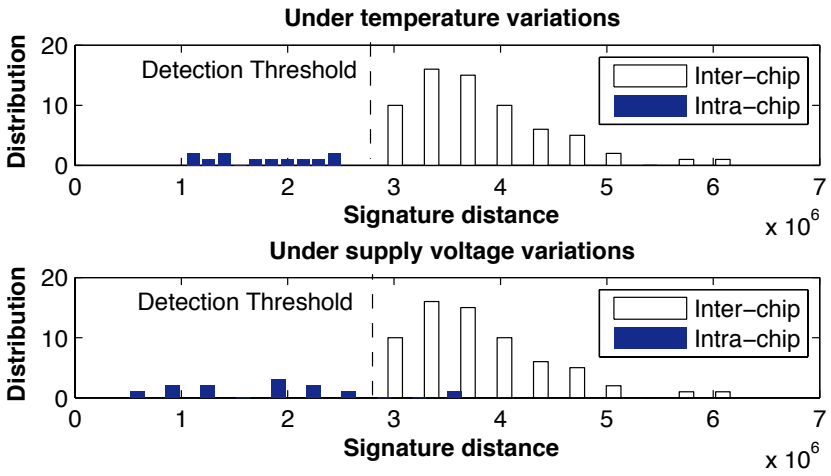


Figure 4.12 : The distribution of the intra- and inter-chip signature L_1 distances

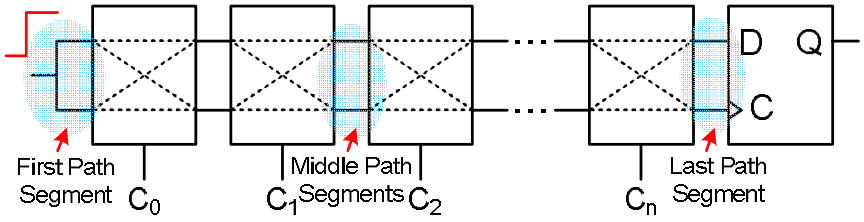


Figure 4.13 : Arbiter-based PUF with path swapping switches.

the first switch, (2) inside the switches, (3) between switches, and (4) after the last switch or before the arbiter (see Figure 4.13). As we will show later, by placing the logic components on symmetric sites and locations on the FPGA, the routing between switches will automatically follow a symmetric route. However, maintaining a *complete* symmetry between the top and bottom path routes before the first switch and after the last switch is structurally infeasible. To alleviate this problem, we introduce and exploit accurate PDLs to tune and remove the bias delay differences caused by asymmetries in net routing. We further introduce a new switch structure that has a symmetric implementation by construction.

4.5.1 Tuning with Programmable Delay Lines

In this section, we introduce a low overhead and high precision PDL with *pico*-second resolution. The introduced PDL is implemented by a single LUT. Figure 4.5 shows the internal structure of an example 3-input LUT. An n-input LUT can be configured to implement any n-input logic function. The LUT in Figure 4.5 is configured so that the inputs A_2 and A_3 act as *don't-care* bits. The LUT output is inverted A_1 and is not a function of A_2 and A_3. However, looking more closely, the inputs A_2 and A_3 determine the signal propagation path inside LUT. For instance, if $A_2A_3 = 00$, the signal propagates through the solid path (red), whereas if $A_2A_3 = 11$, the signal propagates through the path marked with the dashed-lines (blue). The lower dashed path is slightly longer than the upper solid path which results in a larger propagation delay.

The Xilinx Virtex 5 FPGA has 6-input LUTs which can implement a PDL with 5 control bits - there are 4 LUTs in each Slice and two Slices per each CLB. Similar to the above example, the first LUT input, A_1, is the inverter input and the rest of the LUT inputs control the delay of the inverter. For, $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=00000, the inverter has the smallest delay (shortest internal propagation path) and for $A_2A_3A_4A_5A_6$=$A_{[2:6]}$=11111, the inverter has the maximum delay. In general if $A_{[2:6]} > A'_{[2:6]}$ then $D_{LUT}(A) > D_{LUT}(A')$, where $D_{LUT}(A)$ and $D_{LUT}(A')$ are the delay of the inverter with A and A' as the control inputs respectively.

We measured the changes in LUTs' propagation delays under different inputs. For delay measurements, we used the timing characterization circuit shown in Figure 4.1. The characterization circuit consists of a *launch* flip-flop, *sample* flip-flop, and *capture* flip-flop, an XOR gate, and the *Circuit Under Test (CUT)* whose delay is to be measured.

At the rising edge of the clock a signal is sent through the CUT by the launch flip-flop. At the falling edge of the clock, the output of the CUT is sampled by the sample flip-flop. If the signal arrives at the sample flip-flop well before sampling takes place, the correct value is sampled. The XOR compares the sampled value with steady state output of the CUT and produces a zero if they are the same. Otherwise, the XOR output rises to '1', indicating a timing violation. If the signal arrival and the sampling time (almost) simultaneously occur, the sample flip-flop would enter into a metastable condition and produce a non-deterministic output. By sweeping the clock frequency and monitoring the rate at which timing errors happen, the CUT delay can be measured with a very high accuracy. For further details on the delay characterization method the reader is referred to [13, 14]. The measurements performed on Xilinx Virtex 5 FPGAs suggest that the maximum delay difference (i.e., A=00000, and A'=11111) achieved by each inverter is $9ps$ on average.

4.5.2 PDL-based Symmetric Switch

The first arbiter-based PUF introduced in [1] (see Figure 4.13) uses path swapping switches as shown in Figure 4.14 (a). The switch, based on its selector bit, provides a straight or cross connection. Figure 4.14 (b) shows the equivalent circuit implementation and delays. The path swapping switch structure does not lend itself to FPGA implementation, since it is extremely difficult to equalize the nominal delays of the top and bottom paths due to routing constraints, i.e., $\boldsymbol{a}$ and $\boldsymbol{d}$ (or the diagonal paths $\boldsymbol{b}$ and $\boldsymbol{c}$). To alleviate the issue, we propose a new non-swapping switch structure as shown in Figure 4.14 (c). The yellow triangles in the figure represent two PDLs. Figure 4.14 (d) shows its equivalent circuit where the nominal delay values of $\boldsymbol{a}$ and $\boldsymbol{d}$ (or the diagonal paths $\boldsymbol{b}$ and $\boldsymbol{c}$) must be the same.

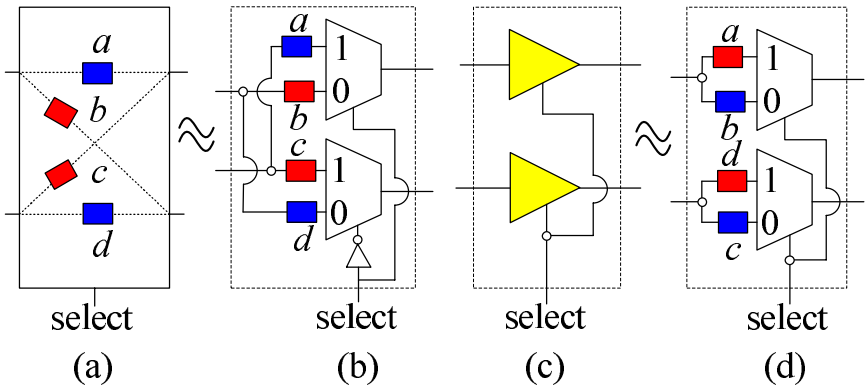


Figure 4.14 : (a),(b) path swapping switch and its delay abstraction (c),(d) PDL-based switch and its delay abstraction.

The complete PUF circuit that uses the new switch structure and the tuning blocks is shown in Figure 4.15. The presented system consists of N switches and K tuning blocks. The tuning blocks insert extra delays into either the top or bottom path based on their selector inputs to cancel out the delay bias caused by routing asymmetry. The only difference between a tuning block and a switch block is that in the former, the selectors to the top and bottom PDLs are controlled independently but in the latter, the same selector bit drives both PDLs. Also note that the tuning blocks do not necessarily have to be placed at the end of the PUF. As a matter of fact, they can be placed anywhere on the PUF in between the switches.

Similar to the arbiter-based PUF with path swapping switches, the new PUF structure is a linear system. The PUF response will be '1' if the sum of the delay switch differences along the path is greater than zero, and '0' otherwise:

$$\sum_{i=1}^{N} C_i \times (a_i - d_i) + (1 - C_i) \times (b_i - c_i) + \Delta \underset{R=1}{\overset{R=0}{\lessgtr}} 0, \tag{4.14}$$

where a_i, b_i, c_i, d_i are the i-th switch delays as shown in Figure 4.14 (d), $C_i \in \{0, 1\}$ is the i-th challenge bit, and R is the response. Also, Δ is a constant delay difference from first and last path segments and tuning blocks lumped together. The security

aspects of the linear PUF structures against machine learning attacks can be boosted by insertion of feed forward arbiter and attaching input/output XOR logic networks to multiple rows of PUFs [2, 57]. The work in analyzing the complexity of machine learning and model attacks against different classes of PUFs [58].

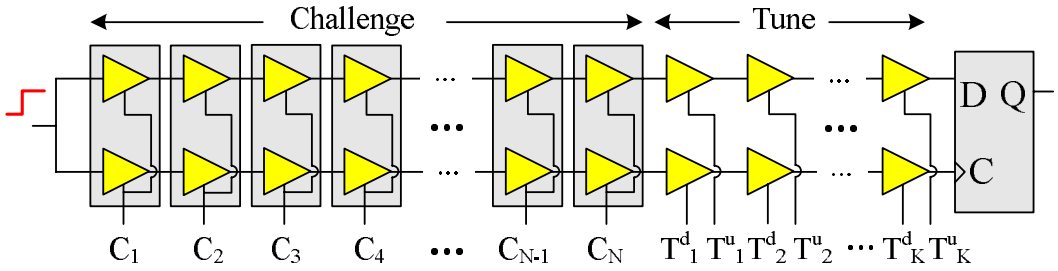


Figure 4.15 : The new arbiter-based PUF structure.

4.6 Precision Arbiter

Arbiters in practice are implemented by D flip-flops. As a result, an arbiter has a limited resolution meaning that if the absolute delay difference of the arriving signals is smaller than its setup and/or hold time, it enters a metastable state where its output becomes highly sensitive to circuit noise and will be unreliable. The probability of flip-flop output being equal to '1' is a monotonically decreasing function of the input signal timing difference (Δ_T). Such probability in fact follows a Gaussian CDF curve as shown in [9, 14]:

$$P_{O=1}(\Delta_T) = Q(\frac{\Delta_T}{\sigma}) \tag{4.15}$$

where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right)$ is the Q function. For an infinitely precise arbiter, σ is infinitesimal i.e. $\sigma \to 1/\infty$, and $P_{O=1}(\Delta_T) \to 1 - U(\Delta_T)$ where U is the step function.

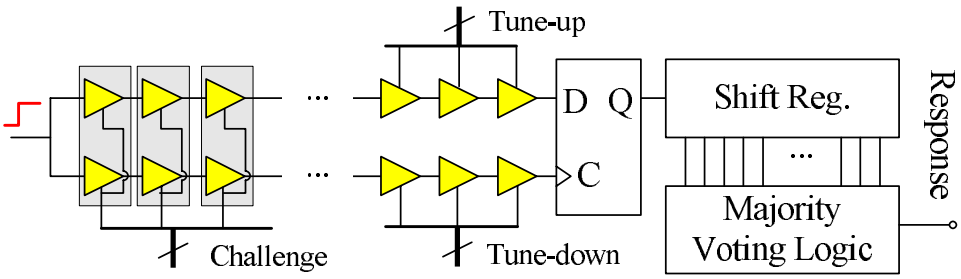


Figure 4.16 : Reducing the response instability due to arbiter metastability by using majority voting.

To increase the arbiter accuracy, we propose multiple evaluations of the same challenge to the PUF and running a majority vote on the output responses as shown in Figure 4.26. The repetitive challenge evaluation combined with majority voting is equivalent to having an arbiter with effectively smaller σ. We will quantify the reduction in σ as a function of the number of repetitions in the experimental results section.

4.7 Robust responses

Fluctuations in operational conditions such as temperature and supply voltage can cause variations in device delays. The impact on delays may not be equal on all devices. As an example, the signal propagation delay on the PUF top and bottom paths is represented in Figure 4.17 by solid and dashed lines respectively. In this example, the path delays increase with temperature at different rates. In the diagram in Figure 4.17 (a), the delay difference Δ_d at the end of the PUF for a given applied challenge at nominal temperature is small, whereas Δ_d in Figure 4.17 (b) is larger for another challenge. The response to the challenge in Figure 4.17 (a) changes as temperature varies because the delays change their order (cross). However, in Figure

4.17 (b) the PUF response remains the same. As demonstrated by this example, the responses to those challenges that cause large delay differences are unlikely to be affected by temperature or supply voltage variations [6].

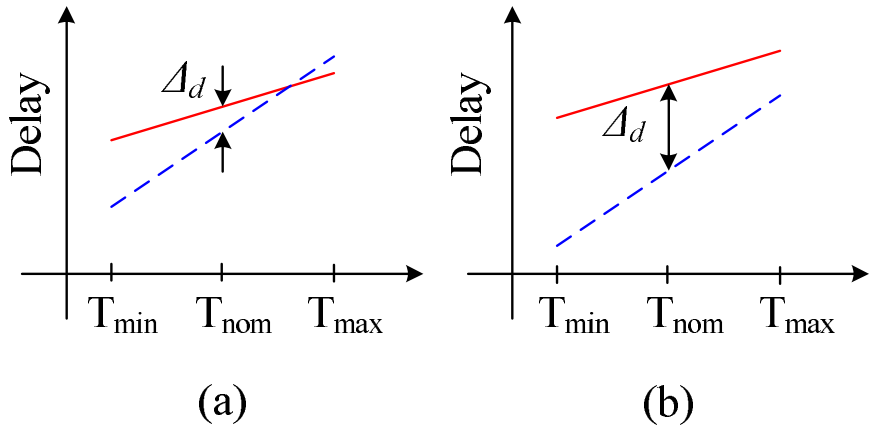


Figure 4.17 : Signal propagation delay as a function of temperature.

In this work, we estimate the delay difference at the input of the arbiter. To estimate the cumulative delay difference (Δ_d), we ought to first train the delay parameters of the linear model of the PUF expressed in Equation 4.14 on the available set of challenge and responses. After estimating the delay parameters, the left hand sum in Equation 4.14 is evaluated for every new challenge. The distribution of the resulting sum (Δ_d) to the set of available challenge-response pairs is next calculated. Now based on the distribution, if the delay difference caused by a given challenge falls in the tails of the distribution, we expect (and will later verify and quantify it through experiments) that the response to this challenge is less likely to be affected by variations in operating conditions. Figure 4.18 shows the distribution of the delay differences at arbiter input to a diverse set of challenges. The challenge set is partitioned into equal sized partitions (bin) based on the delay difference each challenge

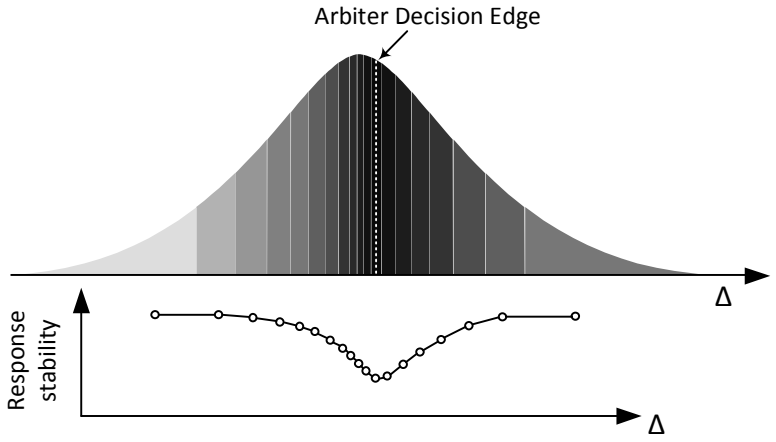


Figure 4.18 : The distribution of Δ_d and stability of responses in the corresponding partitions.

produces. Next, the stability of response to the challenges in each set is measured. We argue that the responses to challenges that fall into the center partitions exhibit lower robustness compared to those in corner partitions.

4.8 Robustness versus Entropy

The next question that arises from classifying robust challenges from non-robust ones is: "Are robust challenges that good?". In other words, are we trading off anything to gain stability and robustness? From information theoretical point of view, it is likely that the responses from more robust challenges bear lower entropy. For example, consider the extreme case where responses are absolutely biased towards either zero or one. In this case we have ultimate robustness whereas the entropy is zero and the responses are not distinct enough for identification. This trade-off (if exists) can only be quantified through measurements. We show this is in fact the case and quantify the loss in entropy in return for robustness in the experimental results section.

4.9 Experimental Evaluation

4.9.1 Programmable delay line

Before moving onto the PUF system performance evaluation, we shall first discuss the results of our investigation on the maximum achievable resolution of the programmable delay lines. We set up a highly accurate delay measurement system similar to the delay characterization systems presented in [13, 12, 14].

The circuit under test consists of four PDLs each implemented by a single 6-input LUT. The delay measurement circuit as shown in Figure A.6 consists of three flip-flops: launch, sample, and capture flip-flops. At each rising edge of the clock, the launch flip-flop successively sends a low-to-high and high-to-low signal through the PDLs. At the falling edge of the clock, the output from the last PDL is sampled by the sample flip-flop. At the last PDL's output, the sampled signal is compared with the steady state signal. If the signal has already arrived at the sample flip-flop when the sampling takes place, then these two values will be the same; Otherwise they take on different values. In case of inconsistency in sampled and actual values, XOR output becomes high, which indicates a timing error. The capture flip-flop holds the XOR output for one clock cycle.

To measure the absolute delays, the clock frequency is swept from a low frequency to a high target frequency and the rate at which timing errors occur are monitored and recorded. Timing errors start to emerge when the clock half period (T/2) approaches the delay of the circuit under test. Around this point, the timing error rate begins to increase from 0% and reaches 100%. The center of this transition curve marks the point where the clock half period (T/2) is equal to the effective delay of the circuit under test.

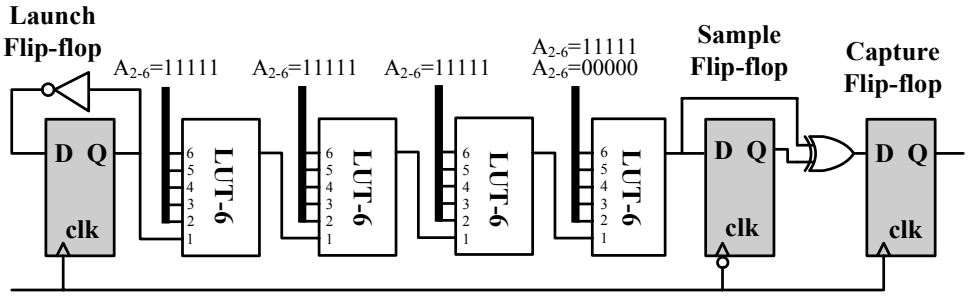


Figure 4.19 : The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL.

To measure the delay difference incurred by the LUT-based PDL, the measurement is performed twice using different (complementary) inputs. In the first round of measurement, the inputs to the four PDLs are fixed to $A_{2-6} = 11111$. In the second measurement the inputs to the last PDL are changed to $A_{2-6} = 00000$. In our setup, a 32×32 array of the circuit shown on Figure A.6 is implemented on a Xilinx Virtex 5 LX110 FPGA, and the delay from our setup is measured under the two input settings. The clock frequency is swept linearly from 8MHz to 20MHz using a desktop function generator and this frequency is shifted up by 34 times inside the FPGA using the built-in PLL.

The results of the measurement are shown on Figure 4.20. Each pixel in the image corresponds to one measured delay value across the array. The scale next to the color-map is in nano-seconds. Figure 4.20 (a) and (b) show the path delay when the last LUT in Figure is driven by $A_{2-6} = 00000$ and $A_{2-6} = 11111$ respectively. Figure 4.20 (c) depicts the difference between the measured delays in (a) and (b). As can be seen, the delay values in (b) are on average about 10 pico-seconds larger than the corresponding pixel values in (a).

4.9.2 Arbiter-based PUF evaluation

Next, we use the programmable delay lines to implement the arbiter-based PUF on FPGA. The implemented PUF has 16 rows whose challenge input bits are connected together and placed in parallel on the FPGA to produce 16 bits of responses per challenge. Each PUF consists of 64 stages of PDLs, where the PDL is implemented by 2 LUTs each acting as an inverter. Figure 4.21 shows the placement and routing of one of the PUF rows. As it can be seen, except for the routing at the beginning and end of the PUF, the rest follows a completely symmetric pattern.

4.9.3 Measurement setup

We have a population of 12 Xilinx Virtex 5 (LX110) FPGAs at our disposal. The FPGAs are mounted on a ball-grid array socket available on Xilinx FF676 Prototype board only. Since the prototype board is stripped of any communication interface, we create a synchronous serial communication protocol to send/receive the data to/from XUP-V5 development board. From the XUP-V5 board, the data is sent to the PC through the Ethernet communication interface at very high speed by using SIRC API. SIRC (Simple Interface for Reconfigurable Computing) is an open sourced software/hardware API developed at Microsoft Research that enables data transfer at full Ethernet speed of 1GB/s between the FPGA and PC [59]. Additionally, to perform measurements under various temperature points, we use PTC10 temperature controller from Stanford Research Systems. The temperature controller drives a TEC (Thermo-electric coupler) Peltier device. TEC is attached on the top the FPGA and beneath a heat-sink. A closed-loop feedback system is established to control the FPGA temperature accurately. The temperature feedback is provided by an on-die diode junction voltage on the Virtex 5 device. This way the stable tem-

perature would be that of the die temperature rather than the package temperature. The temperature controller is further calibrated to reliably map the junction voltage of the diode to die temperature using the temperature readings obtained through ChipScope Pro on-die temperature sensor. The measurement system connections and setup is depicted in Figure 4.22. Figure 4.23 shows the measurement system setup in the lab. The raw data and scripts and software is made available online at `http://aceslab.org/node/1012`.

4.9.4 Tuning the PUF

Before using the PUF, in order to see any changes in the responses, it must be tuned to remove the delay bias resulting from routing asymmetry. In the first experiment, we look at all 16 responses to find out at what tuning level their responses to a set of random challenges are %50 zeros and %50 ones. To be able to find the best tuning level, we feed the PUF with a set of 64,000 random challenges while for each challenge, we sweep the tuning level from -10 to 40. In each sweep point (each tuning level), we collect 64,000 responses from each PUF row (64,000×16 total for each FPGA). Then, we look at the percentage of ones and zeros in each response set across different tuning levels and find the set that is properly balanced.

We refer to *tuning level* as the difference in the number of '1's in the top and bottom PDL selector bits. The tuning level can be either positive or negative indicating insertion of delays to the top and bottom path respectively. Note that when the tuning level is set for example to 40, then it means that 40 of the PDL blocks out of 64 blocks are dedicated to tuning and only 24 bits of the inputs serve as the input challenge.

The response to a given challenge at each tuning level is repeated 128 times, and

a majority vote on the responses is performed to resolve the repeated readings to a single response value. Figure 4.24 shows the ratios of ones in each response set (y-axis) as a function of tuning level (x-axis) for FPGA number 6. Since each PUF on each FPGA produces 16 response bits, there are 16 lines on each subplot. There are 9 subplots in each plot. Each subplot corresponds to the measurement taken under a different operating condition. The center subplot refers to the normal operating condition (i.e. supply voltage V_{DD}= 1 V and room temperature of 30^{o}C). Note that plot is only for one FPGA (FPGA number 6). We have repeated the same experiment on all 12 FPGAs in the lab and the results are available online at [[XXXX.com]].

Figure 4.25 shows the distribution of center of the transition points across all PUFs on all FPGAs.

4.9.5 Majority Voting

As discussed in the work, repeating the challenges to the PUF and running majority voting on the obtained responses can help improve the precision of the arbiter. In this section, we quantify this effect. Figure 4.26 shows the probability of observing a '1' output from a flip-flop as a function of the input signals delay difference. This characteristic has been measured on Xilinx Virtex 5 FPGAs [9, 14]. The width of the transition region (3σ) gets narrower as evaluation is repeated and more statistics is gathered.

The equivalent σ which represents the width of the metastable window (i.e., 3σ) is calculated for different number of repetitions as shown Figure 4.27. The reduction in the metastable window width is logarithmic with respect to the number of repetitions. For 10 repetitions, $\sigma = 2.5\ ps$.

4.9.6 Robust response classification

Next, we measure the effect of robust challenge classification on PUF error rate in presence of temperature and supply voltage variations as discussed in Section 4.3. Each challenge to the arbiter PUF creates a delay difference (Δ) at the input of the arbiter (flip-flop). The Δs produced by all challenges in the challenge space form a Gaussian distribution. If half of the responses are one and half are zero, then this distribution has a mean of zero. The distribution is split by the arbiter decision edge. Those challenges that create a Δ that is larger that e, result in a '1' response and a zero response otherwise, where e is basically the arbiter bias remained after tuning. We partition the Δ distribution and the corresponding challenge space into 20 sets of equal size. The Δs close to the decision border and their corresponding responses are more sensitive to environmental condition fluctuations, and those farther apart from the decision border (i.e. $|\Delta - e| \gg 0$) are less affected by such variations. The Figure 4.28 shows the robustness of the responses to different subset of challenges. The x-axis in each subplot refers to the challenge partition (bin) number. Each partition contains $64000/20 = 3200$ challenges. The y-axis shows the stability of the corresponding responses, where '1' means no errors in the responses and '0' means completely erroneous responses. The error is measured by comparing the responses from eight corner cases to the response at the normal operating condition (room temperature and nominal supply voltage). Therefore, each subplot contains eight lines for each corner case. As it can be observed the challenges in bins that are closer to the decision border produce responses with larger error rates. There are 16 subplots in each figure where each correspond to a PUF output response bit.

Figure 4.29 shows the distribution of the error rates for each challenge partition using boxplots. Each subplot corresponds to an operating condition corner. As it

can be seen, the average error rates is considerably lower at corner (lower and higher) partitions.

4.9.7 Robustness versus entropy

Now that we have quantified the stability of responses to different challenges, it is time to investigate the entropy of the responses to such challenges. In order to quantify the entropy, we look at the inter-chip Hamming distance of PUF responses to challenges in different partitions. For the 12 available FPGAs, 66 distinct pairing of FPGAs can be selected. However, since the tuning level of each PUF on FPGA is different, the challenge set is selected based on the target FPGA. For example, the challenge set for the pair FPGA A and FPGA B is different from FPGA B and FPGA A. This asymmetric challenge selection requirement also means that the inter-chip Hamming distance between FPGA A and FPGA B might be different from FPGA B and FPGA A. Therefore, we investigate the Hamming distance for all 12×11 possible pairing (of course excluding similar chip parings). At each partition, a set of 3200 response vectors of size 16 bits are compared to another set. The result is 3200 integer hamming distances between 0 and 16. We take the average value as the inter-chip hamming distance and normalize it with 16. Next we need to link entropy with Hamming distance. Entropy is maximum if the average normalized inter-chip hamming distance is at 0.5. Any deviation from 0.5 lowers the entropy. In other words, both Hamming distance of 0 and 1 indicate entropy of zero. Figure shows the entropy as measured by Hamming distance for response to challenges in each partition. Each line on this figure corresponds to one paring of FPGAs.

4.9.8 Correlation between effects of temperature and power supply variations

Variation of temperature and/or core voltage from nominal values changes the response to challenges, especially the non-robust challenges. We argue that response flips due to change in temperature is related to response flips due to change in core voltage. Temperature testing is expensive; if a correlation between variation due to temperature and variation due to core voltage can be established even partially, it will help predict temperature effects from core voltage effects and thus lead to a huge cost saving.

The 64000x16 responses for each of the 12 FPGA under various experimental conditions (different temperature and voltage) are used to quantify this argument. The response set obtained in a reference condition is compared to the response set obtained in condition N_1 and the challenges for which the response flips are noted, where N_1 condition being an increment (or decrement) in core voltage from the reference value.

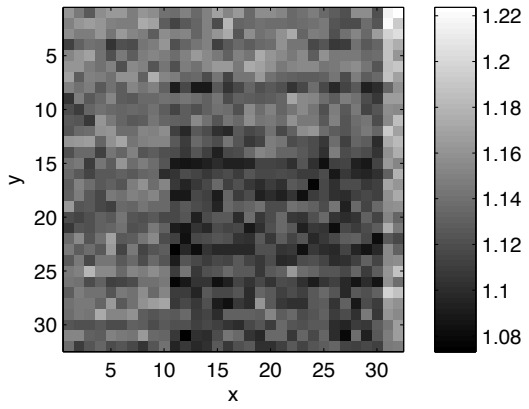
Then the response set obtained in reference condition are compared to the response set obtained in N_2 condition only for the challenges noted in N_1, where N_2 condition being an increment (or decrement) in temperature from the reference value. In other words, if the response to challenge "C", flips (changes from 0/1 to 1/0) as the power supply goes from V_1 to V_2, how likely is it that the response to the same challenge "C", flips as the temperature goes from T_1 to T_2 (while the core voltage stays at V_1). Each PUF is set at a characteristic tuning level for which it has an equal probability of 0 or 1 as an output and the response set is analyzed at that characteristic tuning level to obtain a response error correlation value. (T_1, V_1) and (T_2, V_2) comprise the condition N_1 and N_2 respectively. Figure 4.31 shows the results as boxplot for 18 different experimental conditions tabulated in Table 4.3. The low/high values for

Case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T_1	L	M	L	L	M	L	L	M	L	M	H	H	M	H	H	M	H	H
T_2	M	H	H	M	H	H	M	H	H	L	M	L	L	M	L	L	M	L
V_1	L	L	L	M	M	M	L	L	L	M	M	M	H	H	H	H	H	H
V_2	M	M	M	H	H	H	H	H	H	L	L	L	M	M	M	L	L	L

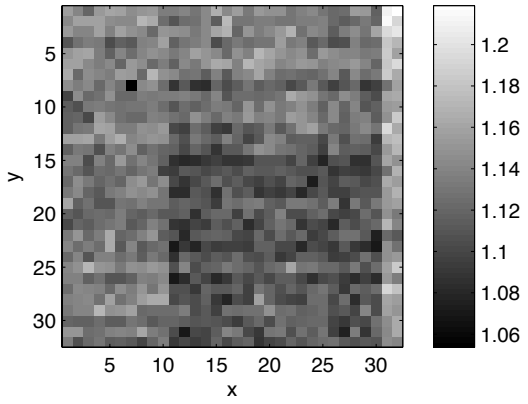
Table 4.3 : 18 correlation cases studies for various increments/decrements on temperature and power supply

core voltage are set assuming a practical tolerance level of 5% in power supply. Low (L), medium (M) and high (H) values for core voltage are 0.95V, 1.00V and 1.05V respectively and for temperature are 5^o C, 35^o C and 65^o C respectively."

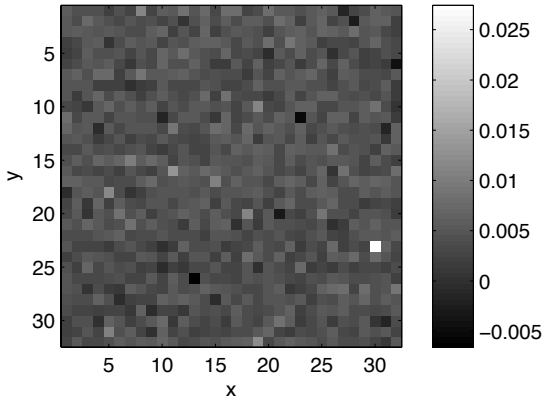
Each box in Figure 4.31 represents the result of the corresponding case and is drawn for the set response error correlation values obtained from 12×16 PUF response sets. The lower and upper edges represent the 25th and 75th percentile respectively while the edge partitioning the box at the centre is the median correlation value from the set of 192 correlation values which is used to quantify this response error correlation. Correlation between voltage and temperature is maximized in case 16 (0.68355), while the correlation in case 7 is also comparable (0.66355). It is interesting to note that case 16 and case 7 are complementary, i.e. (T1, V1) are interchanged with (T2, V2).



(a) Delay for $A_{2-6} = 00000$

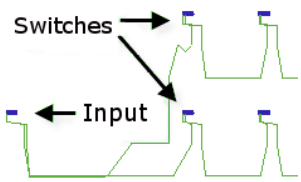


(b) Delay for $A_{2-6} = 11111$

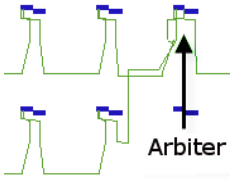


(c) Delay difference

Figure 4.20 : The measured delay of 32×32 circuit under tests containing a PDL with PDL control inputs being set to (a) $A_{2-6} = 00000$ and (b) $A_{2-6} = 11111$ respectively. The difference between the delays in these two cases is shown in (c).



(a)



(b)

Figure 4.21 : Routing and placement of the PUF (a) first segment (b) last segment.

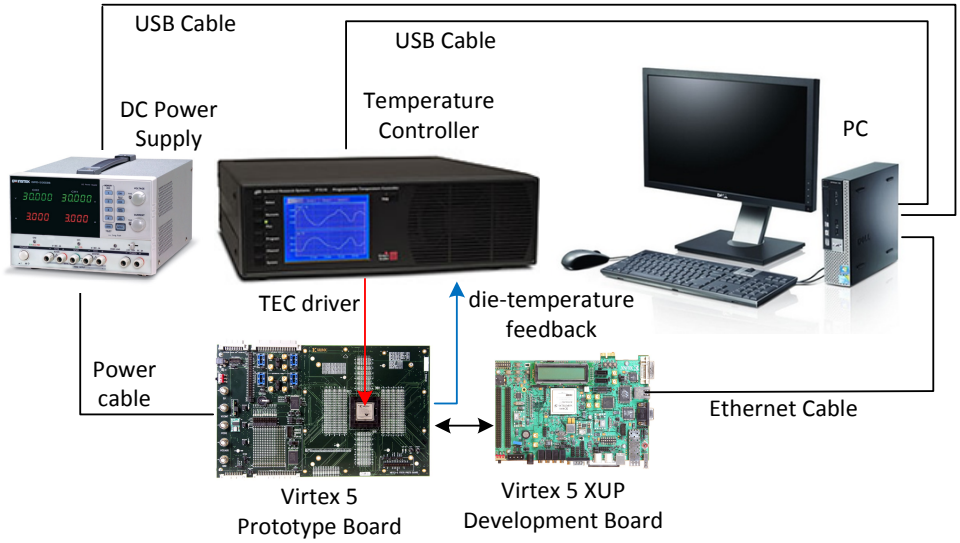


Figure 4.22 : Measurement system setup diagram.

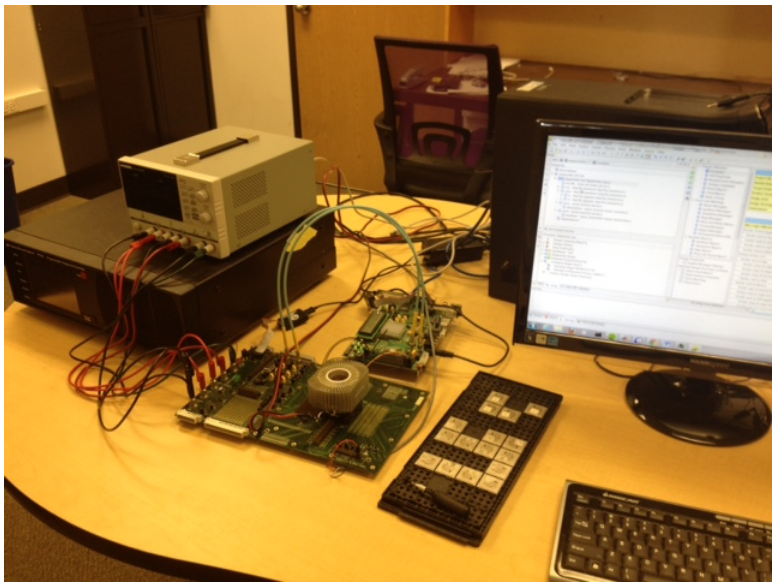


Figure 4.23 : Lab setup.

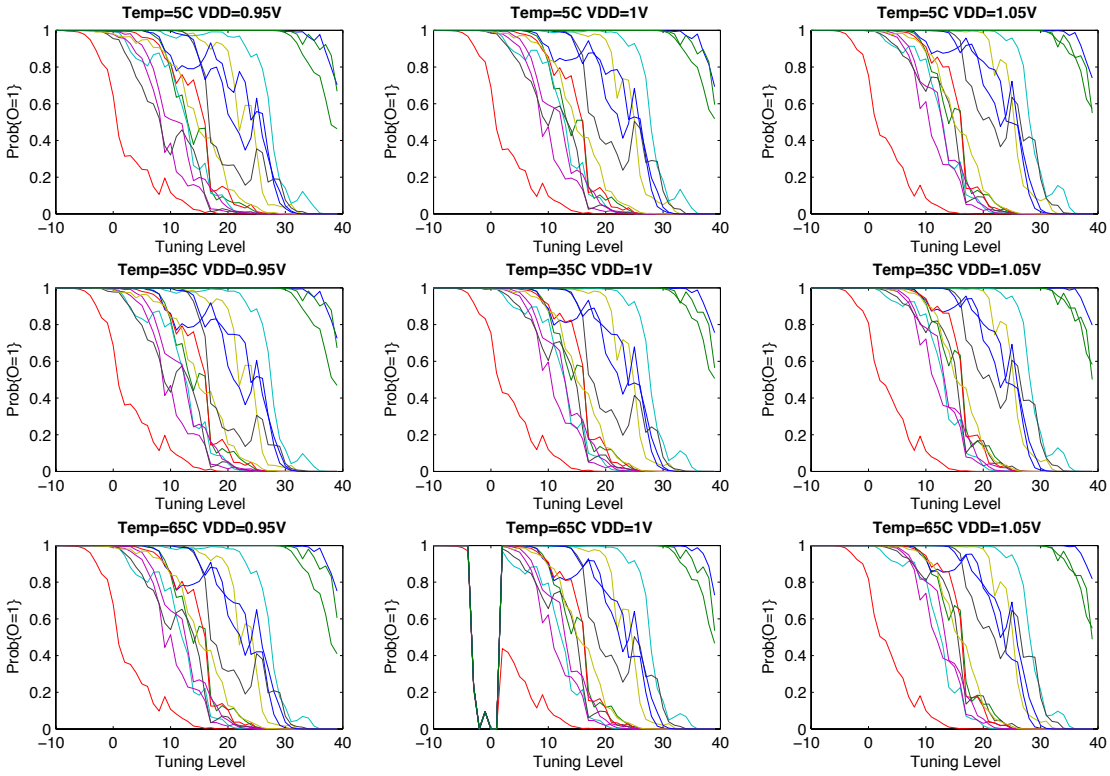


Figure 4.24 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 6.

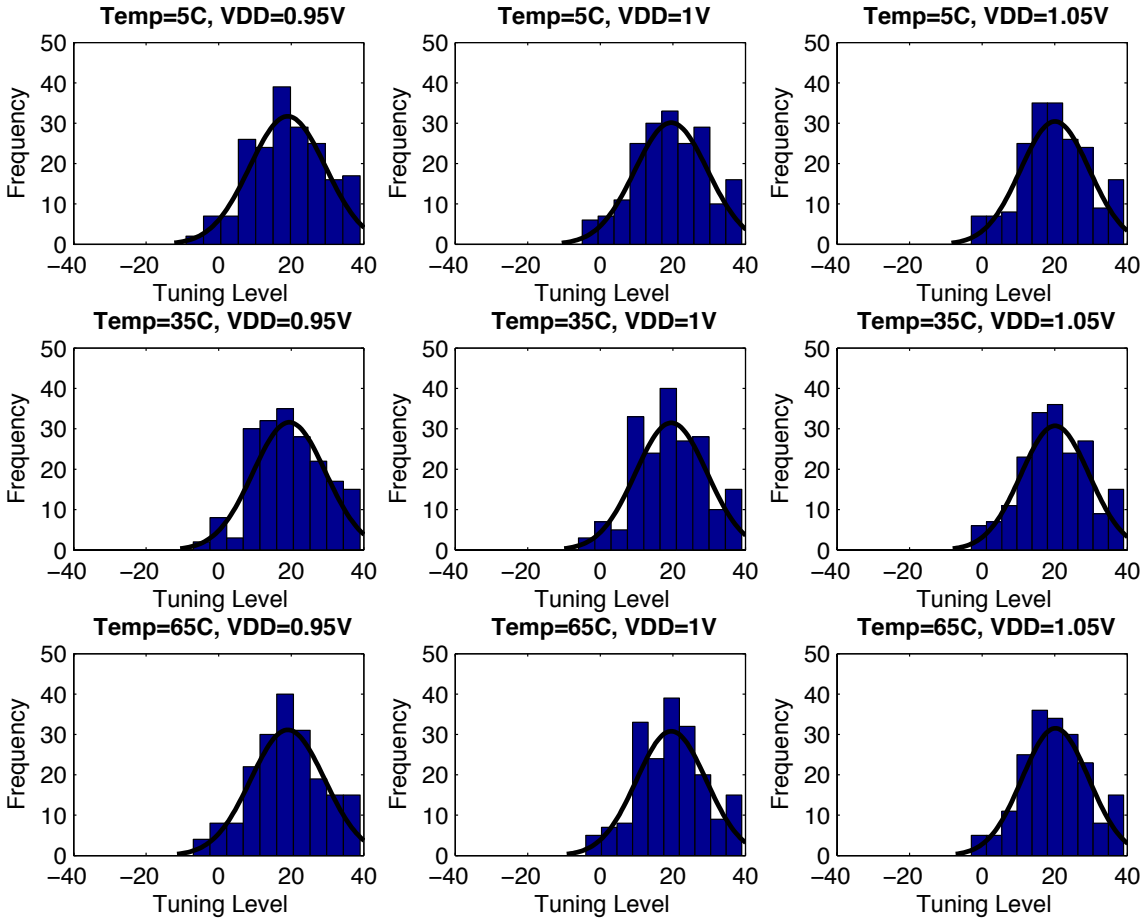


Figure 4.25 : Distribution of the tuning levels across all PUF rows on all FPGAs for different operating conditions.

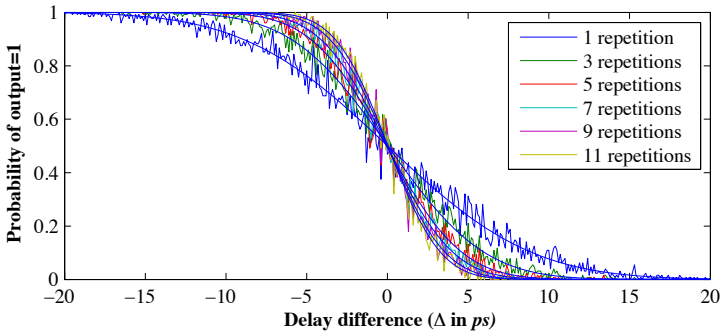


Figure 4.26 : The probability of majority voting system output being equal to 1 as a function of the delay difference.

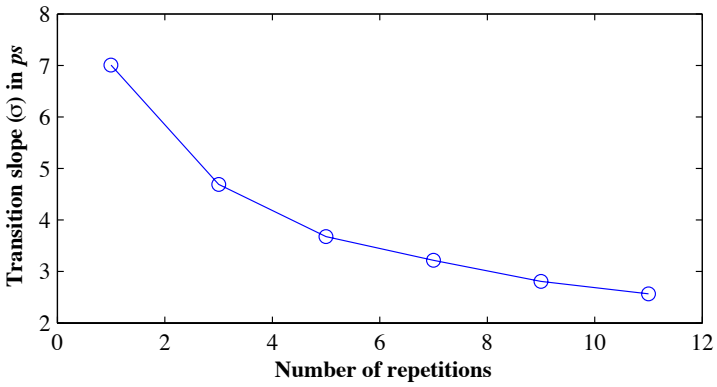


Figure 4.27 : The sharpness (σ) of the transition slope versus the number of repetitions for majority voting.

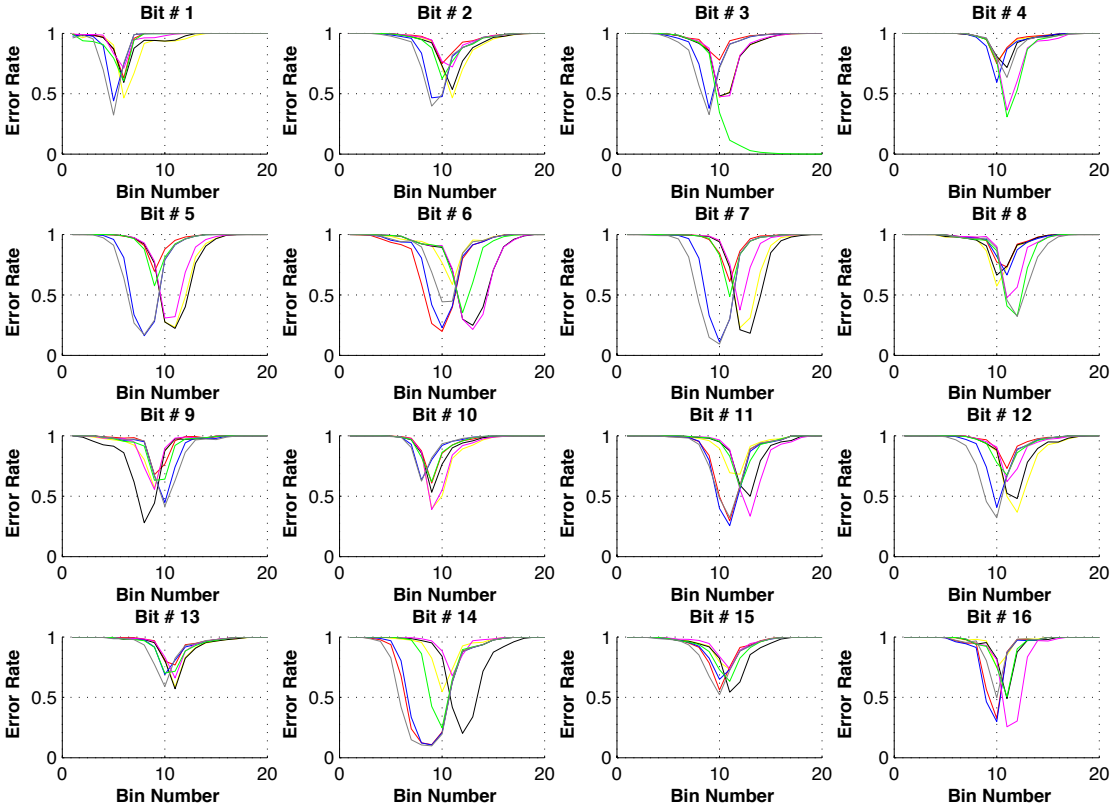


Figure 4.28 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 6.

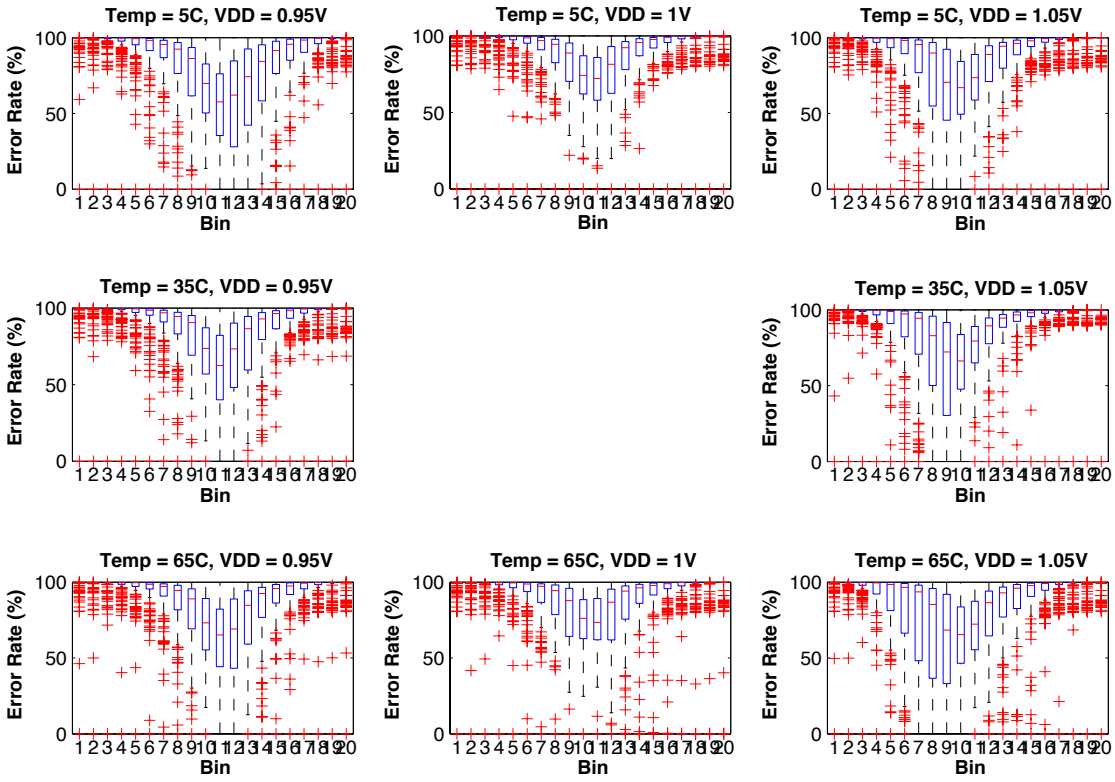


Figure 4.29 : Boxplot showing the distribution of error rates for a given operating condition corner and challenge partition.

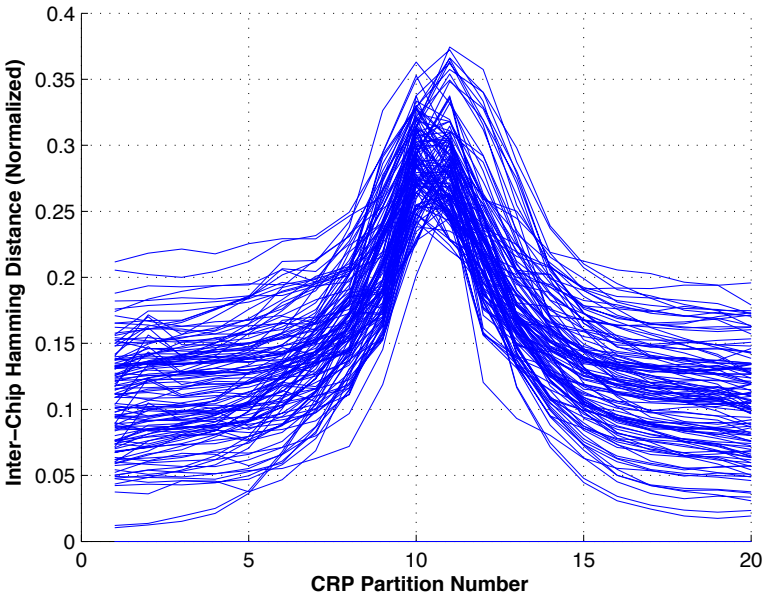


Figure 4.30 : Entropy of the response to the challenges at each robustness partition.

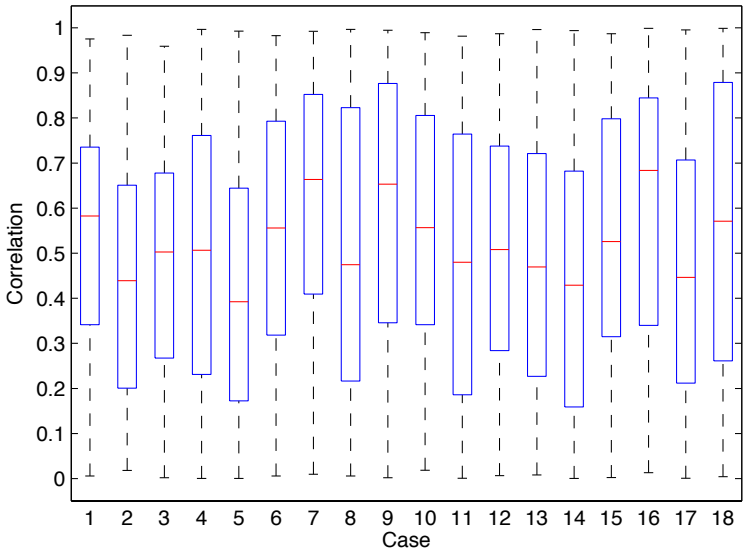


Figure 4.31 : The correlation between effect of temperature and power supply variations on responses for 18 different scenarios. Each box plot is made of response correlation values across 12x16 PUFs.

Chapter 5

PUFs based on current variations

5.1 Concept and circuit realization

In this section, we present the concept and circuit architecture of the new low power current-based PUF. Figure A.1(a) depicts the conceptual architecture of our new PUF circuit. First, process sensitive (PV) voltages/currents are generated. These quantities should ideally be as much sensitive to process parameters as possible but highly insensitive to environmental parameters such as temperature to achieve high levels of response stability and robustness. Next, based on a given input challenge, a subset of these voltages/currents are selected and combined. The combined quantities are compared and converted to digital responses. The comparator maybe tuned for maximum accuracy and reliability based on the predicted statistics of the compared signals.

The circuit implementation of the proposed PUF concept is shown in Figure 5.2. In this implementation, process sensitive currents are generated by using individual FETs whose gate voltages are tied to a fixed voltage source. Next, based on the input challenge that drive the differential current switches, a subset of currents are selected and combined. In other words, by connecting the outputs of the current switches as illustrated in Figure 5.2 and controlling the inputs to the current switches, we can select and add up a subset of currents into either left and right side of the circuit which accordingly flows into the left and right inputs of the sense amplifier. Note

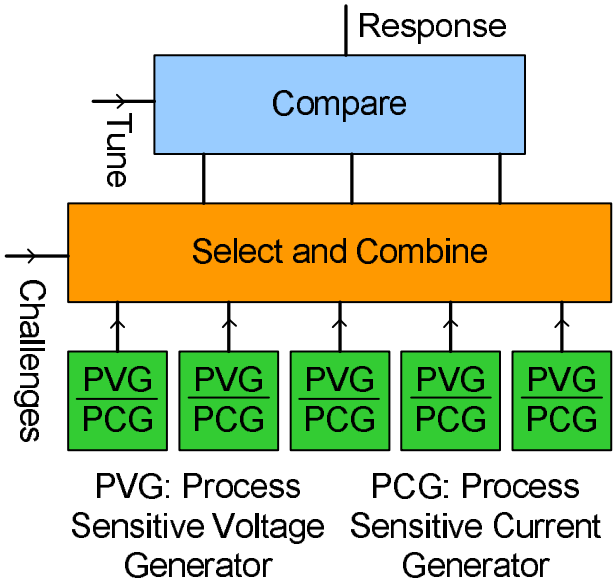


Figure 5.1 : The conceptual block diagram of the proposed PUF structure.

that if both input challenges to a current switch are set to '0' (ground) no current will flow to either left or right sides. Additionally if both input challenge bits are set to '1' (V_{DD}), then half of the total current that enters the current switch will flow through each side. If input challenge bit on one side is '0' and '1' on the other side, the total current that enters the current switch from the bottom single FET current generator will be stirred to the latter side. Equations 5.1 and 2 formally express each current in terms of the inputs to the current switch.

$$I^a[i] = \begin{cases} I[i], & \text{if } C^a[i] = 1 \text{ and } C^b[i] = 0; \\ 0.5I[i], & \text{if } C^a[i] = 1 \text{ and } C^b[i] = 1; \\ 0, & \text{if } C^a[i] = 0 \text{ and } C^b[i] = X; \end{cases} \tag{5.1}$$

$$I^b[i] = \begin{cases} I[i], & \text{if } C^a[i] = 0 \text{ and } C^b[i] = 1; \\ 0.5I[i], & \text{if } C^a[i] = 1 \text{ and } C^b[i] = 1; \\ 0, & \text{if } C^a[i] = X \text{ and } C^b[i] = 0; \end{cases} \tag{5.2}$$

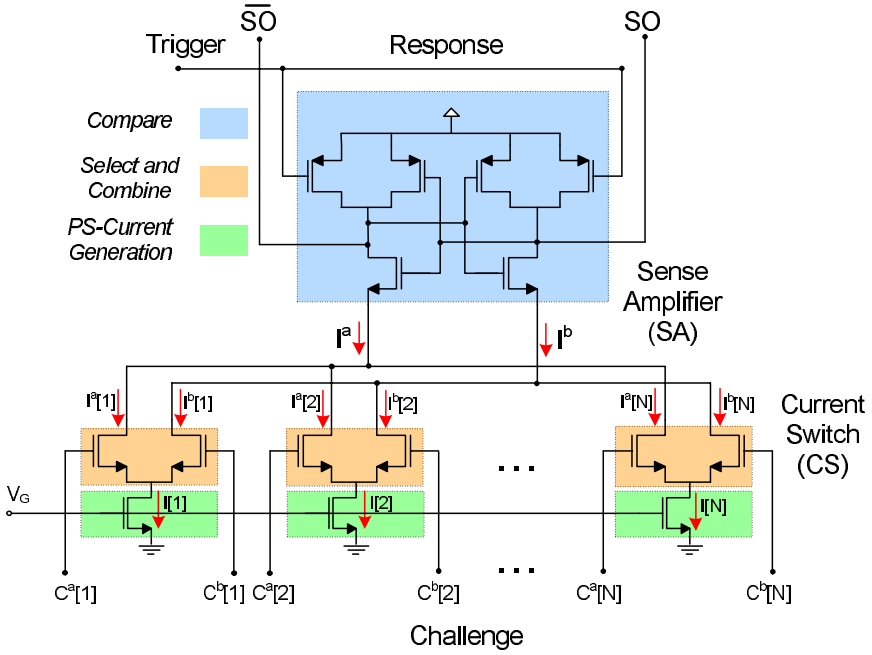


Figure 5.2 : The proposed current based PUF system.

The 'X's in Equation 5.1 and 2 represent 'don't-care'. $I^a[i]$ and $I^b[i]$ denote the left and right output currents of the i-th current switch respectively. Also $C^a[i]$ and $C^b[i]$ respectively represent the left and right inputs to the i-th current switch. Therefore the total current on the left side, i.e. I^a in Figure 5.2, and on the right side, i.e. I^b in Figure 5.2, can be written as the sum of each individual current on each side,

$$I^a = \sum_{i=1}^{N} I^a[i], \qquad I^b = \sum_{i=1}^{N} I^b[i] \tag{5.3}$$

where N is the total number of PV current generator FETs (or current switches). Now, the total current on both sides flows into a latch-based sense amplifier. The

sense amplifier, based on which current is larger, will produce a zero or one digital response, i.e.,

$$\text{response} = \begin{cases} 1, & \text{if } I^a > I^b; \\ 0, & \text{if otherwise;} \end{cases} \tag{5.4}$$

The latch-based sense amplifier used in the PUF system in effect consists of a pair of back-to-back connected inverters. Initially, the output of the inverters are pulled up to V_{DD} by the trigger signal, charging the output node capacitance. Once the challenges are applied, trigger signal goes to zero releasing the output nodes. Soon after the currents start flowing though both sides of the sense amplifier, the output capacitance begin discharging. The discharge pace of the node capacitances is a function of each current magnitude; i.e., the larger the current, the faster the discharge. Whichever node voltage drops first by V_{th} turns on the top inverter transistor and establishes a positive feedback which settles to a response. After the sense amplifier settles, one of the transistors in each inverters turns off and the current flow stops automatically.

In order to avoid any bias and predictability of the output responses and to achieve maximum randomness in responses, the mean/nominal value of the compared currents must be the same. Meeting such property requires the number of combined currents on each side or equivalently the number of ones in the right and left input challenges to be equal, i.e.,

$$\sum_{i=1}^{N} C^a[i] = \sum_{i=1}^{N} C^b[i]. \tag{5.5}$$

In case of existence of any bias in sense amplifier operation, calibration and tuning can be performed by introducing imbalances in Equality 5.5 to have more the number of ones in challenges (larger the nominal current value) on the desired side. This degree of freedom can also be used to sift and distinguish the robust challenges from unstable challenges [16].

5.2 Experimental results

In this section, the evaluation results for the new PUF architecture are presented. We simulate the system with N=64 current generators (and current switches) using IBM 90nm technology models. To achieve maximum level of variability, the device sizes are set to the technology minimum of W/L = 120nm/100nm. Using Monte Carlo simulation guided by the IBM statistical models, 100 circuit instances are generated. Next, we apply 100 challenges to each PUF circuit instance at frequency of 100MHz and the responses to the applied challenges are evaluated and stored. We refer to this setup as the base experiment. In what follows, we run multiple instances of the basic experiment under different scenarios and operational conditions. Figure 5.3 shows the sense amplifier response waveform to a series of random input challenges.

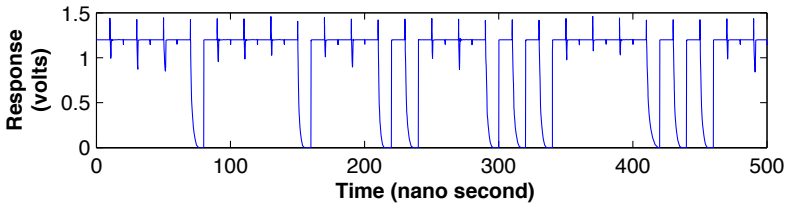


Figure 5.3 : The sense amplifier output response waveform to a set of random challenges.

The first experiment consists of twelve base experiment runs under the combination of the following two sets of scenarios; In the first set of scenarios, the number of active currents on each side is set to 8, 16, 32 (the number of ones in each challenge vector, i.e., $K = \sum_{i=1}^{N} C^a[i] = \sum_{i=1}^{N} C^b[i]$ where $K = \{8, 16, 32\}$). In the second set of seniors, the gate voltage of the current-generator FETs (V_{gate}) is set at different ratios of V_{DD}, i.e. $V_{gate} = \{0.1, 0.3, 0.5, 0.7\} \times V_{DD}$. All of the base experiments are performed under normal operating conditions i.e. temperature of 25^{o}C and V_{DD}=1.2

volts. Therefore, if we define E_1 as the set of scenarios for the first experiment, then $E_1 = \{S_{k,v} \mid (k, v) \in K \times V_{gate}\}$, where $S_{k,v}$ is the experiment scenario under a given k and v.

For each experiment the number of '1's in 100 responses is counted and normalized to 100. Ideally, we would like to have equal number of ones and zeros in responses for highest level of randomness (see [2]). Figure 5.4 shows the distribution of this value across the 100 PUF instances versus different gate voltages for different number of active currents using boxplots. The central mark on the boxplot denotes the median, the edges of the boxes correspond to the 25th and 75th percentiles, the whiskers extent to the most extreme data points and the red plus signs show the outlier points. As it can be observed on the plots, for $V_{gate}/V_{DD} = 0.1$ the responses are highly biased toward '1'. A closer investigation reveals that the for this gate voltage, the generated currents are too small to provoke any response from the sense amplifier.

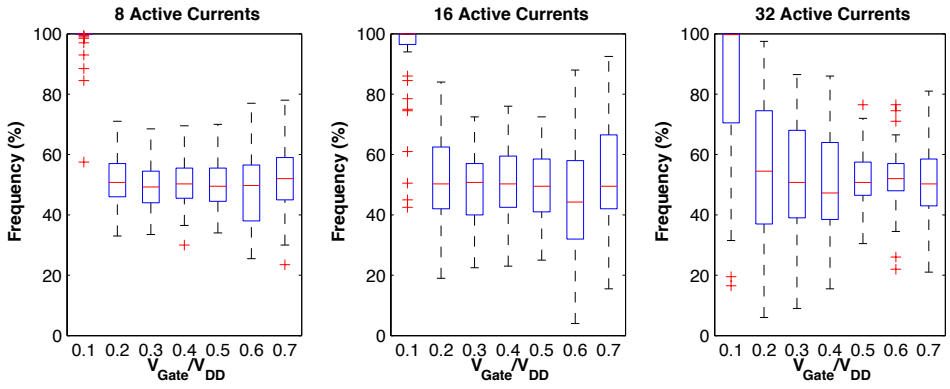


Figure 5.4 : The distribution of number of '1's in responses to 100 challenges over 100 PUFs obtained from pre-layout monte-carlo simulation

The goal of the second and third experiments is to find the operation parameters which achieves the highest level of robustness against the fluctuations in temperature and supply voltage. Similar to the previous experiment, we first define a set

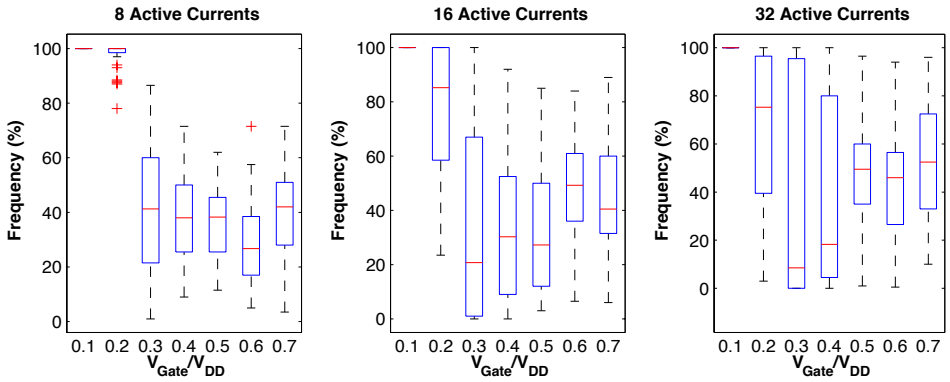


Figure 5.5 : The distribution of number of '1's in responses to 100 challenges over 100 PUFs obtained from post-layout monte-carlo simulation

of scenarios under which we run the base experiment. Let us denote the second set of experiments by E_2 such that $E_2 = \{S_{k,v,t} \mid (k, v, t) \in K \times V_{gate} \times T\}$ where $T = \{-55, 125\}$ are the operating temperatures in Celsius degrees, and K and V_{gate} are the sets defined previously. Next the responses from experiments in scenarios $\{S_{k,v,t} \mid t = -55\}$ are compared to the responses from $\{S_{k,v,t} \mid t = 125\}$ for all k, v and the discrepancies and differences are counted and normalized to the total number of responses (=100). Note that the same challenges are applied to the PUF in each experiment. These two low and high temperatures correspond to standard military operational conditions. The same experiment is repeated for $T = \{-40, 85\}$ and $T = \{0, 75\}$ each corresponding to industrial and commercial operational conditions respectively. The plots on the top row of Figure 5.6 depict the results of this experiment. The 'y' axis on each plot shows the error rate in the responses averaged over the 100 PUF instances and the 'x' axis corresponds to the gate voltage of the current generator FETs. The lines on each plot marked by stars, circles, and dots correspond to commercial, industrial and military operational conditions respectively. The columns in the plot from left to right correspond to the cases where 8, 16, and 32

currents are activated, combined, and compared. As it can be observed, increasing the gate voltage of the current generator FETs raises the response error rates and thus reduces the level of robustness in responses. Moreover, the results suggest that as larger number of currents are combined, the error rate also increases. Note that the error rates for $V_{gate}/V_{DD} = 0.1$ are invalid due to the large bias in the responses as shown in Figure 5.4. The plots in the bottom row of of Figure 5.6 present the same results, however, this time the temperature is fixed to 25^{o}C and supply voltage is varied in three intervals of $V_{DD} = \{1.1, 1.2\}$, $V_{DD} = \{1.1, 1.3\}$, and $V_{DD} = \{1, 1.4\}$. The same conclusions apply to these results as well. Finally, note that the lowest error rate can be achieved for the smallest sub-threshold currents that are large enough to drive the sense amplifier. The PUF consumes 150 μWatt for a duration of 250 ps per each response bit.

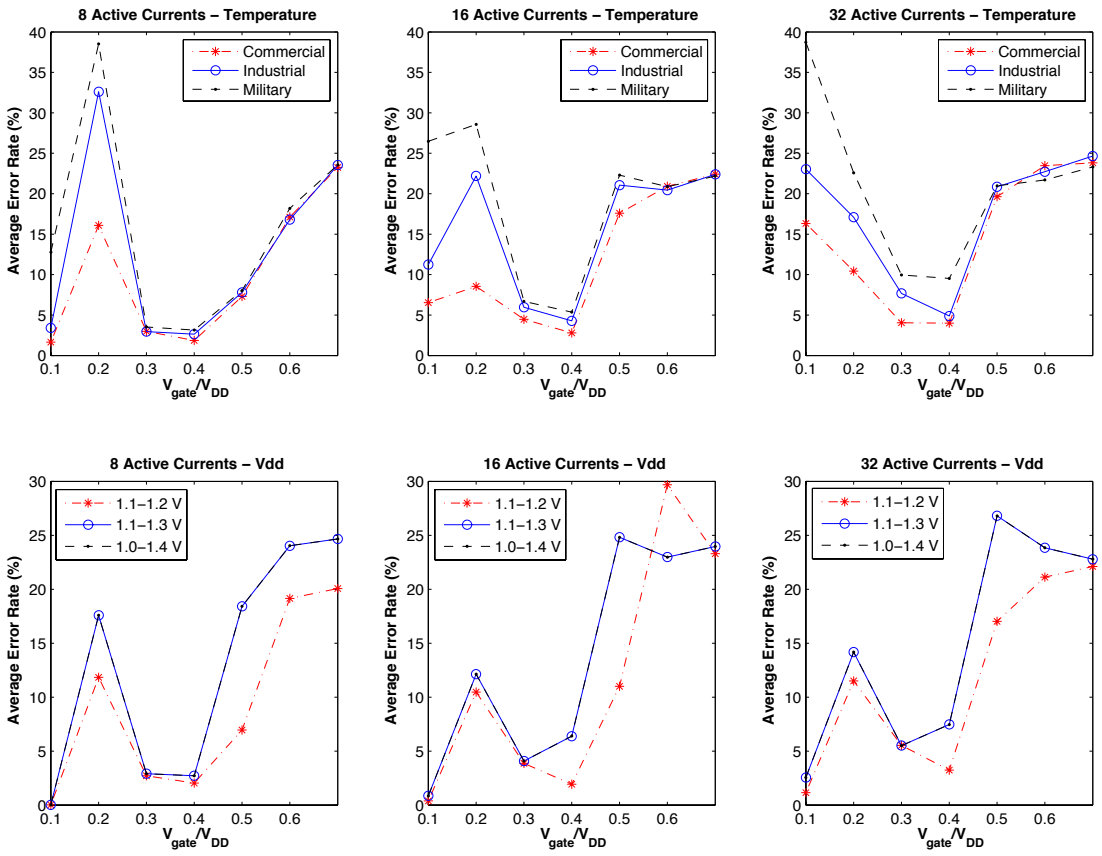


Figure 5.6 : The average response error rate as a function of the current generator transistor gate voltage obtained from pre-layout monte-carlo simulation.

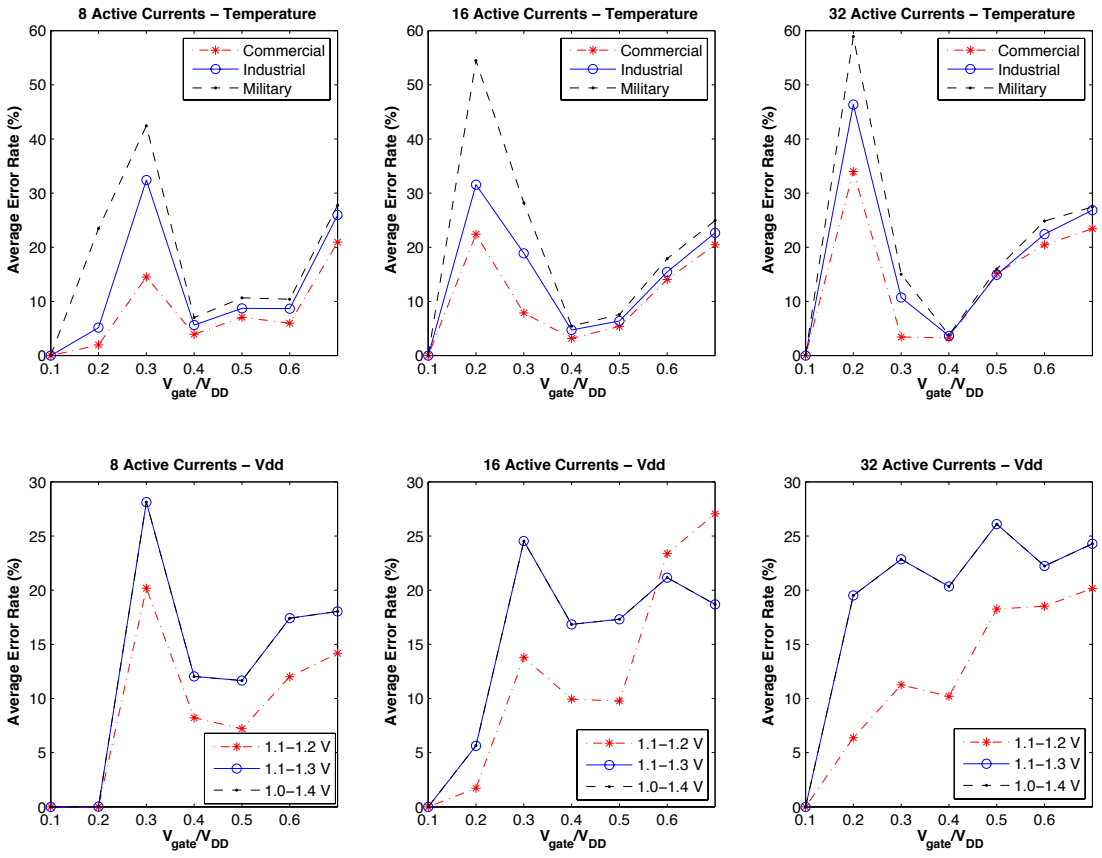


Figure 5.7 : The average response error rate as a function of the current generator transistor gate voltage obtained from pre-layout monte-carlo simulation.

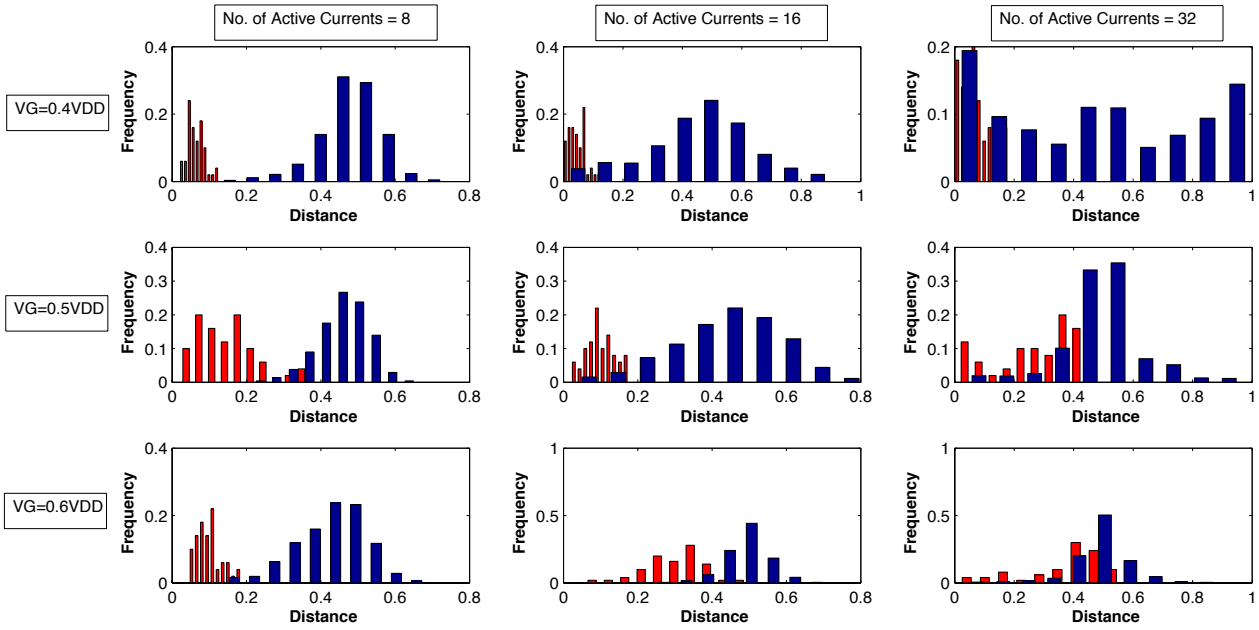


Figure 5.8 : The inter-die and intra-die response distance distribution under different usage scenarios.

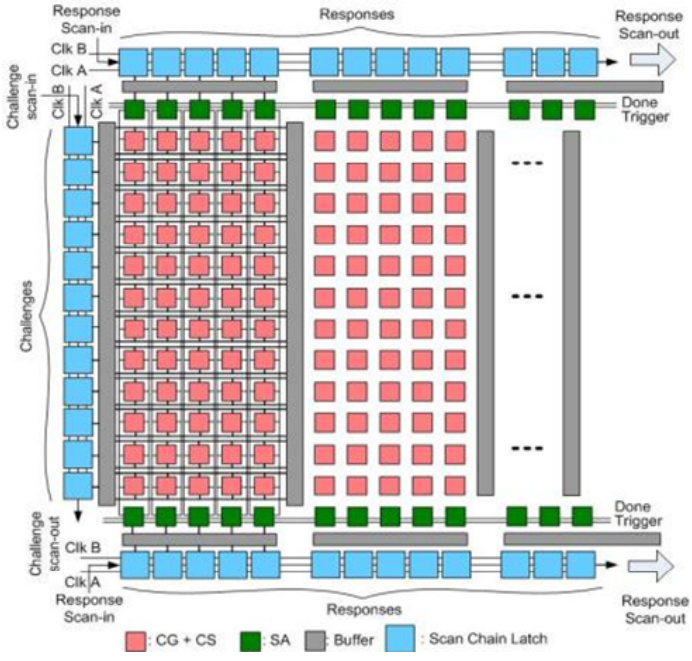
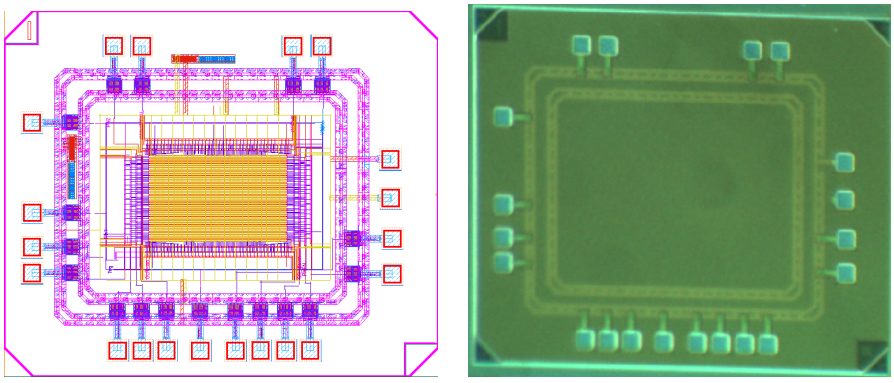


Figure 5.9 : The floor planning of the PUF components on the die.



(a) (b)

Figure 5.10 : (a) The PUF chip layout. (b) taped-out chip micrograph

Chapter 6

Authentication Protocols

6.1 Authentication Protocol

In this section, we show how the extracted cell characteristics in Section 4.1 can be utilized for FPGA authentication. The following terminology is used in the remainder of the thesis. The *verifier* (V) authenticates the *prover* (P) who owns the genuine FPGA device. The verifier authenticates the device by verifying the unique timing properties of the device.

6.1.1 Classic Authentication

The registration and authentication processes for the classic authentication case are demonstrated in the diagram in Figure 6.1(a) and (b) (disregard the darker boxes for now). The minimum required assumptions for this case are: (i) the verifier is not constrained in power, (ii) it is physically impossible to clone the FPGA, and (iii) the characteristics of the FPGA owned by the prover is a secret only known to the prover and verifier.

As shown in Figure 6.1(a), during the registration phase, the verifier extracts and securely stores the cell delay parameters by performing characterization as explained in Sections 4.1. By knowing the FPGA-specific features in addition to the structure and placement of the configured PUF, the verifier is able to predict the responses to any challenges to the PUF. After registrations, the FPGA along with the pertinent

PUF configuration bitstream is passed to the end-user.

At the authentication, end-user (prover) is queried by the verifier to make sure she is the true owner of the FPGA. Classic authentication is shown in Figure 6.1(b). To authenticate the ownership, the verifier utilizes a random seed and generates a set of pseudorandom challenge vectors for querying the prover. The prover responds to the challenges she receives from the verifier by applying them to the configured FPGA hardware. The verifier then compares the received responses from the prover with the predicted ones, and authenticates the chip if the responses are similar.

To ensure robustness against errors in measuring the delays and the changes in operational conditions, the registration entity may also compute the error correction information for the responses to the given challenges. To prevent information leakage via the error correction bits, secure sketch techniques can be used. A secure sketch produces public information about its input that does not reveal the input, and still permits exact recovery of the input given another value that is close to it [49].

The device is authenticated if the response after error correction would be mapped to the verifier-computed hash of responses. Otherwise, the authentication will fail. Alternatively, the verifier can allow for some level of errors in the collected responses and remove the error correction and hashing from the protocol. However, accepting some errors in the responses makes the verifier be more susceptible to emulation/impersonating attacks [1, 2].

6.1.2 Time-bounded Authentication Using Reconfigurability

After the FPGA registration, the verifier is able to compute and predict the responses to any set of challenges by knowing (i) the cell-level features of the pertinent FPGA, (ii) the circuit structure, and (iii) placement of the PUF circuit. The information

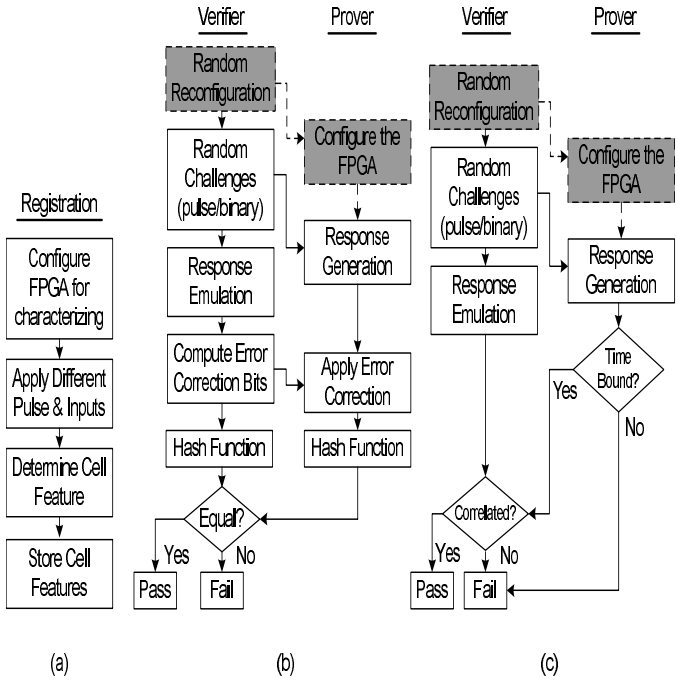


Figure 6.1 : (a) FPGA registration (b) Classic authentication flow (c) Time-bound authentication flow.

on the PUF circuit structure and placement is embedded into the configuration bitstream. In the classic authentic method, the bitstream is never changed. A dishonest prover, off-line and given enough time and resources can (i) extract the cell-level delays of the FPGA, and (ii) reverse engineer the bitstream to discover the PUF structure and its placement on the FPGA. During the authentication, the dishonest prover can compute the responses to the given challenges online by simulating the behavior of the PUF on the fly and producing the responses that pass the authentication.

A stronger set of security protocols can be built upon the fact that the prover is the only entity who can compute the correct response to a random challenge within a specific time bound since he has access to the actual hardware. In this protocol, prior to the beginning of the authentication session, the FPGA is blank. The verifier then sends a bitstream to the device in which a random subset of LUTs are configured for

authentication. After the device is configured, the verifier starts querying the FPGA with random challenges. The verifier accepts the responses that are returned back only if $\Delta t \leq \Delta t_{\max}$ where Δt is the time elapsed on the prover device to compute the responses after receiving the configuration bitstream, and $\Delta t_{\max}$ is the upper bound delay estimated computation of responses by the authentic FPGA prover device, which is composed of device configuration, response generation, error correction, and hashing time all performed in hardware.

The verifier will authenticate the device only if the time the device takes to generate the response is less than $\Delta t_{\max}$. We denote the minimum emulation time by $t_{\min}^{\text{emu}}$, where $t_{\min}^{\text{emu}} >> \Delta t_{\max}$. Time-bounded authentication protocol can be added to the authentication flow, as demonstrated in Figure 6.1(c). Compared to the classic authentication flow, a time bound check is added after the hash function. While performing the above authentication, we emphasize on the assumption that the time gap between hardware response generation and simulation (or emulations) of the prover must be larger than the variation in the channel latency. The time-bound assumption would be enough for providing the authentication proof [9, 30, 60].

Estimating the time-bound

Now let us look at Δt, the time elapsed on the prover device to compute the responses. Before proceeding, note that the characterization is a one-time offline operation which happens prior to authentication phase and its time complexity does not affect the time-bound discussed here. Δt is the sum of time required to configure the FPGA, T_{conf}, and the time spent on evaluating the PUF, T_{eval}, i.e., $\Delta t = T_{conf} + T_{eval}$. During the PUF evaluation, N_p clock pulses at N_f distinct frequencies are sent to N_{cell} PUF cells in serial with an average pulse width of T_{avg}, therefore the average evaluation

time is, $T_{eval} = N_p \times N_{cell} \times N_f \times T_{avg}$. For instance, in our experiments, $N_p = 8$, $N_p = 6$, $N_{cell} = 1024$, and $T_{avg} = 2ns$, yielding $T_{eval} \simeq 98\mu$seconds.

Configuration time varies for different configuration schemes and depends on the configuration file size, configuration port width, and frequency of the driving clock. Configuration time can roughly be estimated by $T_{conf} = \frac{L_b}{f_c \times P_w}$, where L_b is the configuration bitstream length in bits, f_c is the clock frequency in Hz, and P_w is the configuration port width in bits. For example, In our experiment on Xilinx Virtex 5 FPGAs (LX110), L_b=3.5MB, f_c=50MHz and P_w=16bit, the configuration time equals 350 milli-seconds. Faster clocks can expedite the configuration process.

6.2 Attacks and Countermeasures

Perhaps the most dangerous attack to an authentication system is impersonation attack. Impersonation attack aims at deceiving the verifier to get through the authentication by reverse-engineering and simulation of the authentic device behavior, or storing and replaying the communication, or random guessing. Storage and reply attacks are impractical as long as the verifier uses a new random challenge every time. Random guessing and prediction attacks pose a threat if the responses have a low entropy and are predictable. As we mentioned in Section 4.2, by setting the input clock pulse widths to the statistical median of the center of transition regions, the entropy of the responses can be maximized. For a fixed binary challenge, there are not more than six independent input clock pulse widths to be tried. In other words, the responses to other input clock pulse widths would lack sufficient entropy. To obtain more response bits, more binary challenges must be used instead.

Among the aforementioned threats, the reverse engineering and simulation attacks are the most critical attacks to address. The time-bounded protocol discussed

in Section 6.3 is constructed based on secrecy in placement of the PUF and the connection of the input challenges to the CUTs. The secret expires within the given time bound. To provide the correct response to a new challenge, the adversary has to reverse engineer the bitstream to decipher the placement and connection of the input challenges to the PUF. Next, he has to simulate (or emulate) the PUF behavior using the public timing characterization. These two steps must be performed within the given time constraint. Even after many years of research in rapid simulation technologies for hardware design and validation, accurate simulation or emulation of a hardware architecture is extremely slow compared to the real device. In addition, even though bitstream reverse-engineering have partially been performed on some FPGAs [61], performing it would require a lot of simulations and pattern matching. Thus, it would take many more cycles than the authentic hardware where the verifying time is dominated by the bitstream configuration time (in the order of 100 mili seconds).

6.3 Slender PUF Protocol

In this section, the proposed protocol is introduced and explained in detail. The protocol is based on a Strong PUF with acceptable statistical properties, like the one shown in Fig. 6.2. The protocol enables a prover with physical access to the PUF to authenticate itself to a verifier. It is assumed that an honest verifier has access to a compact secret model of the relationship between Strong PUF challenge-response pairs (CRPs). Such a model can be built by training a compact parametric model of the PUF on a set of direct challenge responses pairs. As long as the PUF challenge response pairs are obtained from the linear PUF, right after the arbiter, building and training such a compact model is possible with a relatively small set of CRPs as

demonstrated in the previous literature [40, 42, 43, 2, 44]. The physical access to the measurement points should be then permanently disabled before deployment, e.g., by burning irreversible fuses, so other entities cannot build the same models. Once this access point is blocked, any physical attack that involves de-packaging the chip will likely alter the shared secret.

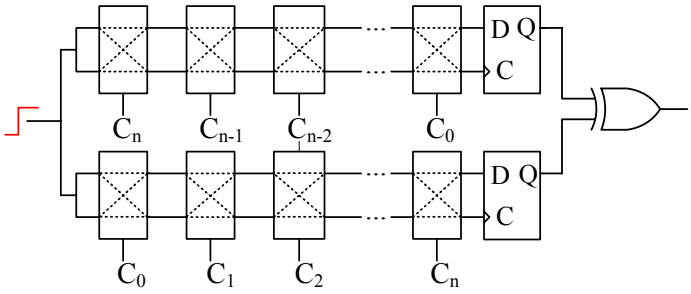


Figure 6.2 : Two independent linear arbiter PUFs are XOR-mixed in order to implement an arbiter PUF with better statistical properties.

The Slender PUF methodology is different from the original PUF challenge response pair identification and authentication methodology. The Slender PUF methodology is devised such that both prover and verifier jointly participate in producing the challenges. The joint challenge generation provides effective protection against a number of attacks. Unlike original PUF authentication methods, an adversary cannot build a database of CRPs and use an entry in the database for authentication.

In the next step of the protocol, the prover generates a set of Strong PUF responses corresponding to the jointly generated challenges. After that, the prover selects a random substring of responses from the response super-string, without revealing the location in the response stream and sends it to the verifier. The verifier, with access to the secret compact PUF model, can perform substring matching, within a predefined error threshold, and validate the responses with a very high probability. The prover

gets authenticated if his submitted response substring matches at any location in the simulated response super-string on the verifier side.

6.3.1 Slender PUF protocol steps

Fig. 6.3 illustrates the steps of the Slender PUF protocol. Steps 1-4 of the protocol ensure joint generation of the challenges by the prover and the verifier. In Steps 1-2 the prover and the verifier each uses its own true random number generator (TRNG) unit to generate a nonce. Note that arbiter PUFs can also be used to implement a TRNG [10]. The prover and verifier generated nonces are denoted by $Nonce_p$ and $Nonce_v$ respectively. The nonces are exchanged between the parties, so both entities have access to $Nonce_p$ and $Nonce_v$. Step 3 generates a random seed by concatenating the individual nonces of the prover and the verifier; i.e., $Seed = \{Nonce_{\mathrm{v}} \parallel Nonce_{\mathrm{p}}\}$.

The generated $Seed$ is used by a pseudo-random number generator (PRNG) in Step 4. Both the prover and the verifier have a copy of this PRNG module. The PRNG output using the seed, i.e., $C = G(Seed)$, is then applied to the PUF as a challenge set (C). Note that in this way, neither the prover nor the verifier has full control over the PUF challenge stream. In Step 5, the prover applies the challenges to its physical PUF to obtain a response stream (R); i.e., $R = \mathrm{PUF}(C)$. An honest verifier with access to a secret compact model of the PUF (PUF_model) also estimates the PUF output stream; i.e., $R^{'} = \mathrm{PUF_model}(C)$.

Let us assume that the full response bitstring is of length L. In Step 6, the prover randomly chooses an index (ind) of bit-size $\log_2(L)$ that points to a location in the full response bitstring. The index is used to generate a substring W from the PUF output bitstream with a predefined length, denoted by L_{sub}. We use the full response string in a circular manner, so that if the value $(ind + L) > \log_2(L)$, the remainder

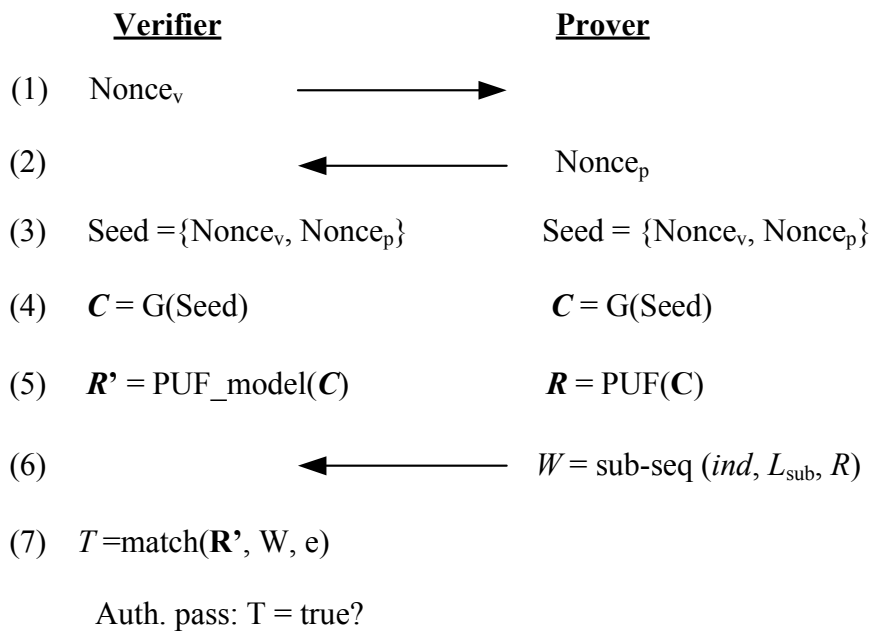


Figure 6.3 : The 7 steps of the SlenderPUF lightweight protocol.

of the substring values are taken from the beginning of the full response bitstream.

The prover then sends W to the verifier. In step 7, an honest verifier, with access to the compact secret PUF model, finds the secret index by searching and matching the received substring to its simulated PUF output sequence (R'). The authentication is successful, only if the Hamming distance between the received and the simulated substrings is lower than a predefined threshold value. In this way, prover does not reveal the whole response stream and the protocol leaks a minimal amount of information. This process is illustrated in Fig. 6.4.

The SlenderPUF protocol is lightweight and is suitable for ultra-low power and embedded devices. Besides a Strong PUF, the prover only needs to implement one TRNG and one PRNG. The information communicated between the parties is also minimal. In addition to exchanging their respective session nonces, the prover only

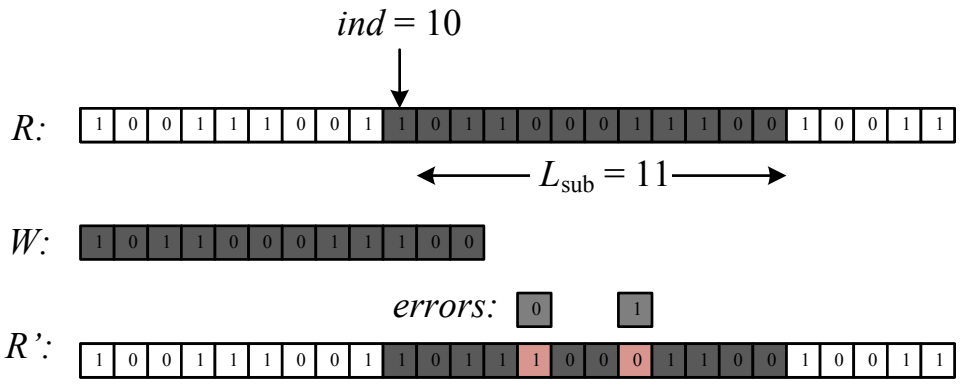


Figure 6.4 : Top: random selection of an index; Middle: extracting a substring of a predefined length; Bottom: the verifier matches the received substrings to its estimated PUF response stream.

needs to send a relatively short substring to the verifier.

6.3.2 Secret sharing

So far we assumed that the verifier possesses a model of the PUF and uses the model to authenticate the prover. The PUF in fact uses an e-fuse to protect the secret and prevent modeling attacks. The chip sets are handled by a trusted party before distributing to end users. The trusted party performs modeling on the PUF and disables the fuse before distribution. Anyone with access to the IC afterwards will not be able to model the PUF since the fuse is disabled. The trusted party can share the PUF models with other authorized trusted parties that want to authenticate the ICs.

The e-fuse mechanism is set up as follows. Before the e-fuse is disabled, the output of the arbiter prior to any XORs can be read and accessed from chip IO pins. This way, the verifier can obtain as many CRPs as needed to build an accurate model of the PUF. After the model is successfully trained, the trusted party and/or the verifier

disables the e-fuse so that no one can obtain the "raw" PUF output.

6.4 Analysis of attacks

In this section, we quantify the resistance of the proposed protocol against different attacks by a malicious party (prover or verifier). First, we quantitatively analyze the resiliency of the method to machine learning and modeling attacks. Second, we probabilistically investigate the odds of authentication by random guessing. Third, we address the attack where a dishonest prover (verifier) attempts to control the PUF challenge pattern. Lastly, the effects of non-idealities of PUFs and PRNGs and their impact on protocol security are discussed.

Throughout our analysis in this section, we investigate the impact of various parameters on security and reliability of protocol operation. Table 6.1 shows the list of parameters.

6.4.1 PUF modeling attack

In order to model a linear PUF with a given level of accuracy, it is sufficient to obtain a minimum number (N_{min}) of direct challenge response pairs (CRPs) from the PUF. Based on theoretical considerations (dimension of the feature space, Vapnik-Chervonenkis dimension), it is suggested in [44] that the minimal number of CRPs, N_{min}, that is necessary to model a N-stage delay based linear PUF with a misclassification rate of ϵ is given by:

$$N_{min} = O(\frac{N}{\epsilon}). \tag{6.1}$$

For example, a PUF model with 90% accuracy, has a misclassification rate of $\epsilon = 10\%$. In the proposed protocol, the direct responses are not revealed and the attacker

Table 6.1 : List of design parameters

Parameter notation	Description
L	Length of PUF response string
L_{sub}	Length of PUF response substring
L_{n}	Length of the nonce
ind	Index value, $0 \leq ind < L$
N_{min}	Minimum number CRPs needed to train the PUF model with a misclassification rate of less than ϵ
k	Number of XORed PUF outputs
N	Number of PUF switch stages
th	Matching distance threshold
ϵ	PUF modeling misclassification rate
p_{err}	Probability of error in PUF responses

needs to correctly guess the secret index to be able to discover L_{sub} challenge response pairs. The index is a number between 0 and $L-1$ (L is the length of the original response string from which the substring is obtained). Assuming the attacker tries to randomly guess the index, then he is faced by L choices. For each *index* choice, the attacker can build a PUF model (M_{index}) by training it on the set of L_{sub} challenge response pairs using machine learning methods.

Now, the attacker could launch L rounds of authentication with the verifier and each time use one of his trained models instead of the actual PUF. If he correctly guesses the index and his model is accurate enough, one of his models will pass

authentication. To build an accurate model as mentioned above, the attacker needs to obtain N_{min} correct challenge response pairs. If $L_{sub} > N_{min}$, then attacker can break the system with O(L) number of attempts. However if $L_{sub} < N_{min}$, then the attackers needs to launch N_{min}/L_{sub} multiple rounds of authentication to obtain at least N_{min} challenge response pairs. Under this scenario, the number of hypothetical PUF models will grow exponentially. Since for each round of authentication there are L models based on the choice of index value, for N_{min}/L_{sub} rounds, the number of models will be of the following order:

$$O(L^{\frac{N_{min}}{L_{sub}}}). \tag{6.2}$$

From the above equation, it seems intuitive to choose small values for L_{sub} to make the exponent bigger. However, small L_{sub} increases the success rate of random guessing attacks. The implications of small L_{sub} will be discussed in more detail in the next section.

The model the attacker is building has to be only more accurate than the specified threshold during the matching. For example, if we allow a 10% tolerance during the substring matching process, then it means that a PUF model that emulates the actual PUF responses with more than 90% accuracy will be able to pass authentication. Based on Equation 6.1, if we allow higher misclassification rate ϵ, then a smaller number of CRPs is needed to build an accurate enough model which passes the authentication.

For example, based on the numbers reported in [44], using 640 CRPs, an arbiter PUF of length 64 can be modeled with an accuracy of 95%. In this example we set the threshold to 5%, then to get an exponent equal to 10 from Equation 6.2, L_{sub} must be 64. In other words, the attacker needs to performs L^{10} operations to obtain 640

CRPs so that he can build a PUF model of 95% accuracy to pass the authentication. For L=1024, L^{10} will be a huge number. However we are faced with another problem. What if the PUF error rate (p_{err}) is higher than the maximum Hamming distance threshold (th)? Then we will have a lot of false negatives (i.e., the honest prover with access to the legitimate PUF will not be able to pass authentication due to noise in responses).

To improve the security while maintaining reliable performance, N_{min} must be increased for a fixed ϵ and N. This requires a structural change to delay based PUF. In this work, we use the XOR PUF circuit shown in Figure 6.2 for two reasons. First, to satisfy the avalanche criterion for the PUF. Second, to increase N_{min} for a fixed ϵ. Based on the results reported in [44], N_{min} is an order of magnitude larger for XOR PUF compared to a simple delay based PUF.

6.4.2 Random guessing attack

A legitimate prover should be able to generate a substring of PUF responses that successfully match a substring of the verifier's emulated response sequence. The legitimate prover must be authenticated by an honest verifier with a very high probability, even if the response substring contains some errors. Therefore, the protocol allows some tolerance during matching by setting a threshold on the Hamming distance of the source and target substrings.

Simultaneously, the probability of authenticating a dishonest prover should be extremely low. These conditions can be fulfilled by carefully selecting the Hamming distance threshold (th), the substring length (L_{sub}) and the original response string length (L) by our protocol.

A dishonest prover without access to the original PUF or its model, may resort

to sending a substring of random bits. In this case, the probability of authenticating a randomly guessing attacker would be:

$$P_{\text{auth,guessing}} \leq L \times \sum_{i=L_{\text{sub}}-th}^{L_{\text{sub}}} \binom{L_{\text{sub}}}{i} \frac{1}{2}^{i} \cdot \frac{1}{2}^{L_{\text{sub}}-i}, \tag{6.3}$$

where L_{sub} and th are the length of the substring and the Hamming distance threshold, respectively. Note that Eq. 6.3 is a binomial cumulative distribution function. For an honest prover, the probability of being authenticated is:

$$P_{\text{auth,honest}} \simeq \sum_{i=L_{\text{sub}}-th}^{L_{\text{sub}}} \binom{L_{\text{sub}}}{i} (1-p_{\text{err}})^{i} \cdot {p_{\text{err}}}^{L_{\text{sub}}-i}, \tag{6.4}$$

where p_{err} is the probability of an error in a response bit. If L_{sub} is chosen to be a sufficiently large number, Eq. 6.3 will be close to zero and Eq. 6.4 will be close to one.

6.4.3 Compromising the random seed

In Slender PUF protocol, the prover and the verifier jointly generate the random PRNG seed by concatenating the outputs of their individual nonces (generated by TRNGs); i.e., $seed = \{Nonce_{\text{v}} \parallel Nonce_{\text{p}}\}$. The stream of PRNG outputs after applying the seed is then used as the PUF challenge set. This way, neither the prover nor the verifier has full control over generating the PUF challenge stream.

If one of the parties can fully control the seed and challenge sequence, then the following attack scenario can happen. A dishonest verifier can manipulate an honest prover into revealing the secret information. If the same seed is used over and over during authentication rounds, then the generated response sequence (super-string) will always be the same. The response substrings now come from the same original

response string. By collecting a large enough number of substrings and putting the pieces together, the original super-string can be reconstructed. Reconstruction will reveal L CRPs. By repeating these steps more CRPs can be revealed and the PUF can be ultimately modeled.

A dishonest prover (verifier) may intentionally keep his/her portion of the seed constant to reduce the entropy of seed. This way, the attacker can exert more control over the random challenges applied to the PUF. We argue that if the seed length is long enough this strategy will not be successful.

This attack leaves only half of the bits in the generated *Seed* changing. For a seed of length $2L_{\mathrm{n}}$-bits (two concatenated nonces of length L_{n}-bits), the chance that the same nonce appears twice is $\frac{1}{2^{L_{\mathrm{n}}}}$). For example, for $L_{\mathrm{n}} = |Nonce_{\mathrm{v}}| = |Nonce_{\mathrm{p}}| = 128$, the probability of being able to fully control the seed will be negligibly small.

Therefore, one could effectively guard against any kind of random seed compromise by increasing the nonce lengths. The only overhead of this approach is a twofold increase in the runtime of the TRNG.

6.4.4 Substring replay attack

A dishonest prover may mount an attack by recording the substrings associated with each used *Seed*. In this attack, a malicious prover records the response substrings sent by an honest prover to an honest verifier for a specific *Seed*. The recording may be performed by eavesdropping on the communication channel between the legitimate prover and verifier. A malicious party may even pre-record a set of response substrings to various random *Seed*s by posing as a legitimate verifier and exchanging nonces with the authentic prover.

After recording a sufficiently large number of *Seeds* and their corresponding re-

sponse substrings, the malicious party could attempt to impersonate an honest prover. This may done by repeatedly contacting the legitimate verifier for authentication and then matching the generated *Seed*s to its pre-recorded database. This attack could only happen if the *Seed*s collide. Selecting a sufficiently long *Seed* that cannot be controlled by one party (Subsection 6.4.2) would hinder this collision attack.

Passive eavesdropping is performed during the pre-recording phase, the chances that the whole *Seed* collides will be $1/2^{L_n}$. The worst-case scenario is when an adversary impersonates a verifier and controls half of the seed which reduces the collision probability to $1/2^{L_n/2}$.

6.4.5 Exploiting non-idealities of PRNG and PUF

Thus far, we assumed that the outputs of PRNG and PUF are ideal and statistically unbiased. If this is not true, an attacker may resort to exploiting the statistical bias in a non-ideal PRNG or PUF to attack the system. Therefore, in this section we emphasize the importance of the PUF avalanche criterion for securing against this class of attacks.

If the PUF has poor statistical properties, then the attacker can predict patterns in the generated responses. The attacker can use these predicted patterns to more confidently find/guess a matching location for the substring. In other words, statistical bias in the responses will leak information about the location index of the response substring.

Recall that an ideal Strong PUF should have the strict avalanche property [9]. This property states that if one bit of the PUF's input challenges is flipped, the PUF output response should flip with a $\frac{1}{2}$ probability. If this property holds, the PUF output for two different challenges will be uncorrelated. Fig. 6.5 shows the

probability of output flipping versus the Hamming distance between two challenge sequences for the Strong PUF proposed in [9]. It is desirable to make this probability as close as possible to $\frac{1}{2}$.

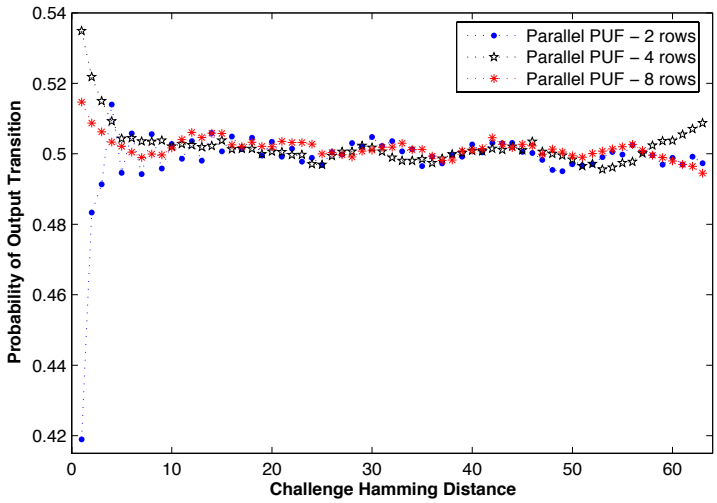


Figure 6.5 : The probability of the arbiter PUF output flipping versus the Hamming distance between two challenge sequences for 2, 4, and 8 independent XOR-mixed PUFs [2]

The figure shows that this probability is very close to the ideal number when at least four independent PUF output bits are mixed by an XOR. As more independent PUF response bits are mixed, the curve moves closer to the ideal case; however, this linearly increases the probability of error in the mixed output bit. For instance, for a single Strong PUF response bit error of 5%, the probability of error for 4-XOR mixing is reported to be 19% in [9].

In our implementation of Slender PUF protocol, Linear feedback shift registers (LFSRs) are used as a lightweight PRNG. An ideal LFSR must have the maximum length sequence property [62]. This property ensures that the autocorrelation function of the LFSR output stream is "impulsive", i.e., it is one at lag zero and is $\frac{-1}{N}$ for

all other lags, where N is the LFSR sequences length. N should be a sufficiently large number, which renders the lagged autocorrelations very close to zero [62]. Therefore, if an LFSR generates a sequence of challenges to the PUF, the challenges are uncorrelated. In other words, for an ideal LFSR, it is highly unlikely that an attacker can find two challenges with a very small Hamming distance.

Even if the attacker finds two challenges with a small Hamming distance in the sequence, Fig. 6.5 shows that the output of our proposed PUF would be sufficiently uncorrelated to the Hamming distance of the input challenges. Therefore, a combination of PRNG and PUF with strict avalanche criteria would make this attack highly unlikely. It is worth noting that it is not required by any means the PRNG to be a cryptographically secure generator. The seed in the protocol is public and the only purpose of the PRNG is to automatically generate a sequence of challenge vectors. Simultaneously, it must not allow an attacker to completely control the challenges and thus the responses.

6.5 Experimental evaluation

In this section, we use the PUF measurement data collected in the lab to estimate the practical protocol parameters values and present methodology to arrive at the parameter values. False acceptance and false rejection probabilities depend on PUF error rates. Unfortunately, there has been no comprehensive reports till this date on PUF response error rates (caused by variations in temperature and power supply conditions) nor any solid data on modeling error rates measured on real PUF challenge response pairs. The data reported in the related literature mainly come from synthetic (emulated) PUF results rather than actual reliable PUF measurements and tests.

We used the data we measured and collected across 12 Xilinx Virtex 5 (LX110)

FPGAs at 9 accurately controlled operating condition (combination of different temperatures and power supply points). Each PUF holds 16 PUFs and each PUF is tested using 64,000 random challenges.

To obtain the error rate, each PUF evaluation at nominal condition (temperature = 35^{o}C and V_{DD} = 1 V) is repeated 128 times and then majority of the response values to the same challenge is taken as the ideal response of PUF. Then the variation from the ideal response is measured compared as the percentage of the bits in the 128-bit vector that deviate from the ideal response. For example if 10 bits from the 128 bits are ones and the rest are zeros, and the deviation from the majority response, or the response error rate, is (10/128)×100 = 7.8%. The same method is used to measure the response error in different operating condition with respect to the ideal response at the nominal condition.

The center cell of Table 6.2 shows the average deviation (taken over 64,000 challenge-response pairs) of these experiments from the ideal response at the nominal condition. These experiments are also repeated for different voltage and temperature conditions and then the average deviation of these outputs from the ideal PUF response is reported in Table 6.2. As it can be seen from this table, the error rate can substantially increase in non-nominal conditions. The worst case scenario happens when the temperature is 5^{o}C and the voltage is 0.95V. The table shows that 30^{o}C degree change in temperature will have a bigger effect on the error rate than a 5% voltage change.

As mentioned earlier, the verifier repeatedly tests the PUF in the factory to obtain a consensus of the PUF responses for an array of random challenges. The verifier then uses the reliable response bits to build a PUF Model for himself. When the PUF is deployed in the field, the prover challenges its own PUF and send the responses to the verifier. The average error rate of the prover response in different working

V_{DD} \ Temperature	5^oC	35^oC	65^oC
0.95 V	8.4%	6.2%	7.1%
1.00 V	6.8%	3.1%	6.4%
1.05 V	7.2%	6.7%	7.9%

Table 6.2 : Average bit error rate of PUF in different voltage and temperature conditions in comparison with the ideal PUF output at nominal condition.

V_{DD} \ Temperature	5^oC	35^oC	65^oC
0.95 V	13.2%	10.5%	10.7%
1.00 V	8.9%	6.4%	8.9%
1.05 V	9.3%	10.2%	11.8%

Table 6.3 : Average bit error rate of the Verifiers PUF model against the PUF outputs in different voltage and temperate conditions.

condition against the verifiers model is listed in Table 6.3. The listed errors are the superposition of two types of error. The first type is the error in PUF output due to noise of environment as well as operating condition fluctuations. The second type is the inevitable modeling error of the verifiers PUF model. These error rates are tangibly higher than the error rates of Table 6.2. The worst error rate is recorded at 5^oC temperature and voltage of 0.95V. This error rate is taken as the worst-case error rate between an honest verifier and an honest Prover. We will use this error rate to estimate the false acceptance and false rejection probability of the authentication protocol.

Q: As explained in the thesis, the attack complexity depends exponentially on the minimum required number of challenge response pairs (CRPs), i.e., N_{min}, to reach a modeling error rate of less than 'th', the matching threshold in the protocol. The matching threshold in the protocol is incorporated to create a tolerance for errors in the responses caused by modeling error as well as errors due to environment variations and noise.

By relaxing the tolerance for errors in the protocol (i.e., increasing 'th'), we basically increase the probability of attack. For example, if we allow for 50% error in during the matching, then any randomly generated response set will pass the authentication. On the contrary, by lowering the tolerance for errors, rate at which authentication of genuine PUF fails due to noisy responses increases. As a rule of thumb, the tolerance has to be set greater than the maximum response error rate to achieve sensible false rejection and false acceptance probabilities.

Once the tolerance level (th) is fixed to achieve the desired false rejection and false acceptance probabilities, N_{min} must be increased to hinder modeling attacks. However,N_{min} and th are inter-related for a given PUF structure. In other words, for a given fixed PUF structure, increasing th mandates that a less accurate model can pass the authentication, and that model can be trained with few number of CRPs (smaller N_{min}). The only way to achieve a higher N_{min} for a fixed th is to change the PUF structure.

Earlier in the thesis, we proposed using XOR PUFs instead of a single arbiter-based PUF in order to increase N_{min} for a fixed th. As reported previously in the related literature, xor-ing the PUF outputs makes the machine learning more difficult and requires a larger CRP set for model building. The major problem with XORing the PUF outputs is error propagation. For example, if the outputs of two arbiter-

Temperature / V_{DD}	5^oC	35^oC	65^oC
0.95 V	24.7%	19.9%	20.3%
1.00 V	17.0%	12.4%	17.0%
1.05 V	17.7%	19.4%	22.2%

Table 6.4 : 2-input XOR.

based PUFs are mixed with XORs, the XOR PUF response error rate will be the sum of each individual arbiter-based PUFs (minus the multiplication of both error). This means the error tolerance has to be also doubled to have a reliable operations. This observation of trade-off between N_{min} and th, led us to quantify this effect. It was nor clear in a practical scenario adding to the number of XORs will make it more difficult to break the protocol or not, since the data reported in other papers are mainly from synthetic (emulated) PUF results rather than reliable PUF error rate estimates.

In order to quantify the trade-off between N_{min} and th, we first calculate the effective compound error rate at the XOR PUF output for different operating conditions and different number of XOR inputs. Tables 6.4, 6.5, 6.6 show the effective response error rate for 2-input, 3-input, 4-input XOR PUF respectively.

According to the above tables, the maximum error rates measured from the XOR PUF responses are 24.7%, 34.6%, 43.2% 2-input, 3-input, 4-input XOR PUF respectively. To guarantee reliable authentication at all operating conditions, the error tolerance th must be set above the maximum error rates. Now after deriving the error tolerance level for each PUF, we would like to know how many challenge response pairs are required to train the PUF model and reach a modeling error that

Temperature / V_{DD}	5^oC	35^oC	65^oC
0.95 V	34.6%	28.3%	28.8%
1.00 V	24.4%	18.0%	24.4%
1.05 V	25.4%	27.6%	31.4%

Table 6.5 : 3-input XOR.

Temperature / V_{DD}	5^oC	35^oC	65^oC
0.95 V	43.2%	35.8%	36.4%
1.00 V	31.1%	23.2%	31.1%
1.05 V	32.3%	35.0%	39.6%

Table 6.6 : 4-input XOR.

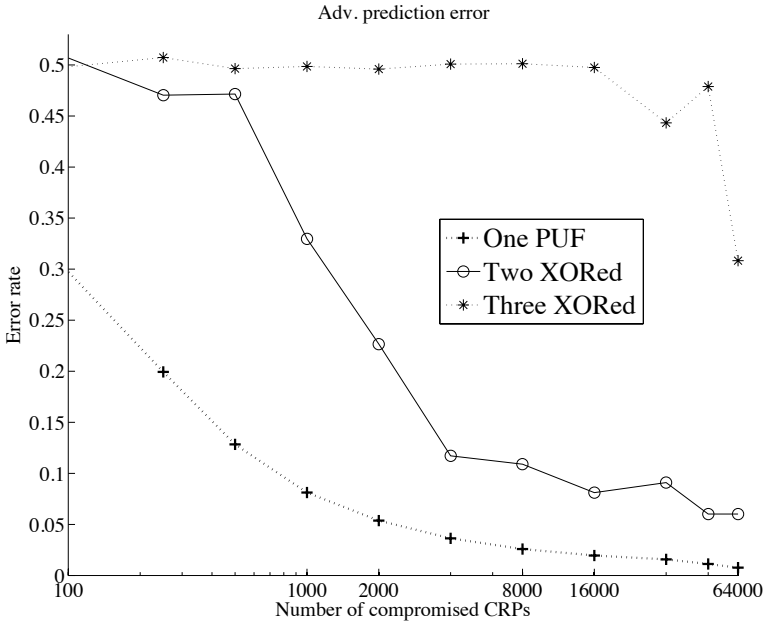


Figure 6.6 : The modeling error rate for arbiter-based PUF, and XOR PUFs with 2 and 3 outputs as a function of number of train/test CRPs.

falls below the tolerance level. In other words, how many challenge/response pairs does the adversary need to collect in order to pass the authentication and break the system?

To answer this question, we trained and tested the PUF model on the data collected in the lab from real PUF implementations. We measured the modeling accuracy as a function of train/test set size for each PUF. The results in Figure 6.6 show the modeling error using evolutionary strategy (ES) machine learning method.

Based on the results in Figure 6.6, the largest N_{min} for all three PUFs, after taking into account the error threshold (th) derived earlier, is achieved for XOR-PUF with 3 inputs, i.e. to achieve a modeling error rate of less than 34.6%, 64,000 CRPs must be collected, therefore N_{min} = 64,000 for 3-input XOR PUF.

Table 6.7 shows the false rejection and false acceptance error rate of our protocol

L_{sub}	500		
Error threshold	210	200	190
False rejection	0.1%	1%	9%
False acceptance	7.1%	1.5%	0.001%
L_{sub}	1000		
Error threshold	395	385	375
False rejection	0.2%	1%	5%
False acceptance	1.1e-8	1.1e-10	8e-13
L_{sub}	1250		
Error threshold	487	477	467
False rejection	0.2%	1%	5%
False acceptance	1.8e-12	0	0

Table 6.7 : False rejection and acceptance error probabilities for different protocol parameters.

with the length of substring (L_{sub}) fixed at 500, with different matching distance thresholds. False rejection rate is the rate in which the service to the truthful prover is disrupted. The requirements on the false rejection rate are not usually as stringent on the requirements on the false acceptance rate, however, one should assume that a customer would deem a product impractical if the false rejection rate is higher than a threshold. In our protocol design, we tune the system parameter to achieve a false negative rate of 1%, while minimizing the false acceptance rate. Also, we take the worst-case error rate as the basis of our calculation for false acceptance and false rejection rates. The error rates that we report are the upper bound of what can be observed in the field by a customer/prover.

Table 6.7 shows that the desired false rejection rate of 1% is achieved when the threshold is 200 for L_{sub}=500. However, this results in a false acceptance rate of 1.5%, which is not acceptable. Therefore, the substring length should be increased in order to achieve a much lower false acceptance probability. Table 6.7 lists these rates for $L_{sub} = 500$, $L_{sub} = 1000$, $L_{sub} = 1250$. The second column for $L_{sub} = 1250$ shows that a false negative rate of 1% and a false positive rate of 0 can be achieved if error threshold is 477/1250 = 38%. If we set L = 1300, with this parameters, an adversary needs to perform $O(1300^{(64000/1250)}) \approx \mathrm{O}(2^{527})$ machine learning attacks in order to break this system which makes the system secure against all computationally bounded adversaries.

6.6 Hardware implementation

In this section, we present an FPGA implementation of the proposed protocol for the prover side on Xilinx Virtex 5 XC5VLX110T FPGAs. Since there is a stricter power consumption requirement on the lightweight prover, we focus our evaluation

on prover implementation overhead. The computation on the verifier side can run solely in software, however, the computation on the verifier may also be carried out in hardware with negligible overhead.

For the Slender PUF protocol, it is desirable to use a low overhead PUF implementation, such as the one introduced in [7]. If an ASIC or analog implementation of the PUF is required, the ultra-low power architecture in [10] is suitable for this protocol. A very low-power verifier implemented by a microcontroller such as MSP430 can easily challenge the PUF and run the subsequent steps of the protocol.

We use the implementation of the arbiter-based PUF in [16]. The arbiter-based PUF on FPGA is designed to have 64 input challenges. In total, 128 LUTs and one flip-flop are used to generate one bit of response. To achieve a higher throughput, multiple parallel PUFs can be implemented on the same FPGA.

There are various existing implementations for TRNGs on FPGAs [63, 56]. We use the architecture presented in [7] to implement a true random number generator. The TRNG architecture is shown in Figure 6.8. This TRNG operates by enforcing a metastable state on the flipflop through a closed loop feedback system. The TRNG core consumes 128 LUTs that are packed into 16 CLBs on Virtex 5. In fact, the TRNG core is identical to the arbiter-based PUF except that the switches act as tunable programmable delay lines. The core is incorporated inside a closed-loop feedback system. The core output is attached to a 12-bit counter (using 12 registers) which monitors the arbiter's metastability. If the arbiter operates in a purely metastable fashion, the output bits become equally likely ones and zeros. The counter basically measures and monitors deviations from this condition and generates a difference feedback signal to guide the system to return back to its metastable state. The counter output drives an encoding table of depth 2^{12} where each row contains a 128-bit word

resulting in a 64KByte ROM. A table of size $2^{12} \times 8$-bits (=4KByte) implemented by a RAM block is used to gather and update statistics for online post processing.

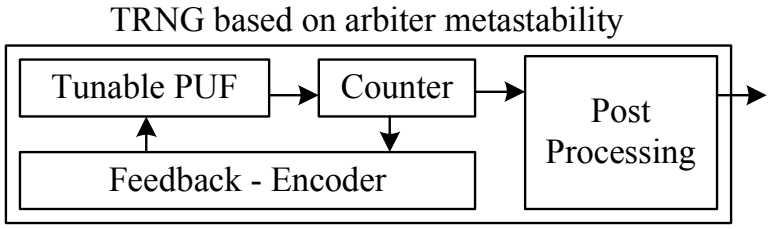


Figure 6.7 : True random number generation architecture based on flipflop metastability

The nonce size is set to 128 for both the prover and verifier. Each 128-bit nonce is fed into a 128-bit LFSR. The content of the two LFSRs are XORed to form the challenges to the PUF.

The pattern selection can be achieved by shifting the intended substring of the PUF responses into a FIFO. The shifting operation, however, begins only when needed. In other words, before running the PUF, the random index is generated by the TRNG. For example, in our implementation the response sequence has a length of 1024 which results in a 10-bit index. To generate a 10-bit random index, we have to run the TRNG 8×10 clock cycles according to Table 6.8. Since we do not care about the response bits that are generated before and after the substring window, we do not need to even generate or store those bits. Therefore, the PUF has to only be challenged for the response bits in the substring. This significantly reduces the overall run time and the storage requirement on the FIFO. The FIFO size is accordingly equal to the length of the substring which is set to 256 in our implementation.

The propagation delay through the PUF and the TRNG core is equal to 61.06ns. PUF outputs can be generated at a maximum rate of 16Mbit/sec. Post-processing

on the TRNG output bits can lower the throughput from 16Mbit/sec to 2Mbit/sec. Since the TRNG is only used to generate the nonce and the index, its throughput does not affect the overall system performance; the number of required true random bits is smaller than the PUF response bits.

Table 6.8 : Implementation overhead on Virtex 5 FPGA

No.	Type	LUT	Registers	RAM blocks	ROM blocks	Clock Cycles
4	PUF	128	1	0	0	1
1	TRNG	128	12	4KB	64KB	8
1	FIFO	0	256	0	0	N/A
2	LFSR	2	128	0	0	N/A
1	Control	12	9	0	0	N/A
Total		652	278	4KB	64KB	N/A

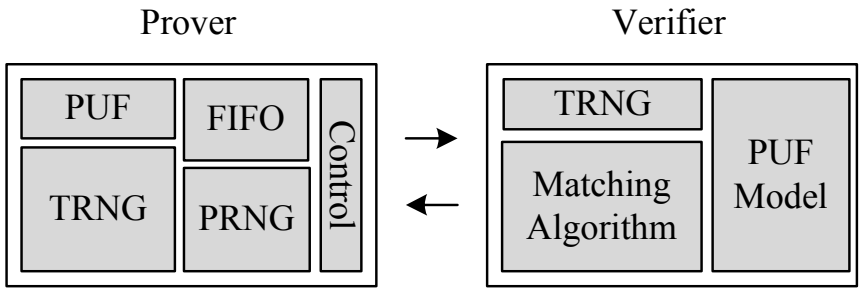


Figure 6.8 : Resource usage on prover and verifier sides

The implementation overhead of our proposed protocol is much less than traditional cryptographic modules. For example, robust hashing implementation of SHA-2

as listed in Table 6.9 requires at least 5492 LUTs of a Virtex-II FPGA [3] and it takes 68 clock cycles to evaluate. This overhead will occur on the top of the clock cycles required for PUF evaluation.

Finally, note that most of the area overhead for the protocol implementation is coming from the TRNG. By using non-volatile memory storage, TRNG can also be avoided. A Slender PUF implementation without a TRNG generates $log_2(L)$ extra response bits and uses the extra bits as the index value. Also to generate the nonce, previously used and revealed substring response bits can re-write the nonces and be used for the next round of authentication. This is because for a statistically unbiased PUF, the responses follow random number properties. In this case, the contents of the non-volatile memory is publicly available and there will be no external access point to change or re-write the values to the memory. However, this implementation is vulnerable to invasive attacks that aim to alter the memory content.

Table 6.9 : SHA-2 implementation overhead as reported in [3]

SHA-256	Freq. (MHz)	Clock Cycles	TP (Mbps)	Area (LUTs)
Basic	133.06	68	1009	5492
2x-unrolled	73.97	28	996.7	8128
4x-unrolled	40.83	23	908.9	11592

Chapter 7

Conclusion

This doctoral thesis presented novel formal requirements, properties, and protocols for PUFs that are used to design new architectures and implementations. The thesis lays out the foundation to formally define and derive a set of requirements and properties for physically unclonable functions. These requirements provide us with tools and guidelines to analyze, test, and evaluate PUF architectures and the implications of the choice of architecture on security and performance.

Once the requirements and desired properties are determined, robust and efficient PUFs architectures were introduced and implemented across various platforms including digital and analog ICs that conform to the introduced requirements. In particular, two PUF implementation were shown on FPGAs leveraging delay variation of digital components. The first method uses an at-speed characterization mechanism to measure component delays. The second is the long-sought implementation of arbiter-based PUF. Many efforts made by the research community to implement the arbiter-based PUF on FPGA had been previously unsuccessful. The main reason for such difficulty was the inability to achieve a symmetric routing of the arbiter PUF. The difficulty arises from the lack of freedom in routing on FPGA dictated by the rigid fabric of FPGA interconnects. In this thesis, I showed the first implementation of arbiter-based PUF on FPGA realized through a novel delay tuning mechanism of pico-second resolution. An ultra-low power analog implementation on ASIC was presented that exploits variations in sub-threshold leakage currents of MOS devices. This

is the most power efficient and smallest PUF known to date. The circuit was taped out in IBM 90nm low power technology. The results show that the leakage-based PUF circuit consumes 40 femto-joules to generate one bit of response. Full performance analysis and comparison were carried out on these implementations. Statistical properties and performance metrics such as response error rate in presence temperature and voltage supply variations as well as speed, area, and power consumption were measured and reported.

Finally, design of low overhead and secure protocols using PUFs was presented. The goal of these protocols is to protect the PUF against machine learning attacks and prevent eavesdroppers or dishonest provers to pass the authentication without having access to the physical medium (PUF). Also, the protocols prevent an attacker disguised as a verifier to extract information from the PUF. The protocols are designed with elegant simplicity in mind specifically to lower overhead and to refrain from using computationally expensive classic cryptographic modules and error correction techniques. Two specific protocols, one exploiting a time bound on the authentication process and the other one utilizing a pattern matching index-based authentication on PUF responses were introduced to integrate the PUF in lightweight applications. The pattern matching protocol use a true random number (TRNG) to generate the nonces and the random secret index. A TRNG based on flip flop metastability and a closed loop feedback system is further developed and implemented on FPGA.

Appendix A

TRNG

A.1 TRNG System Design

To drive the flip-flop into its metastable state, we use an at-speed monitor-and-control mechanism that establishes a closed loop feedback system. The monitor module keeps track of the output bit probabilities over repeated time intervals. It then passes on the information to the control unit. The control unit based on the received probability information decides to add/subtract the delay to/from top/bottom paths to calibrate the delay difference so that it gets closer to zero. For instance, if the output bits are highly skewed towards 1, then the delay difference (Δ) must be decreased by increasing the top path delay to balance the probabilities. Figure A.1 (a) demonstrates this concept.

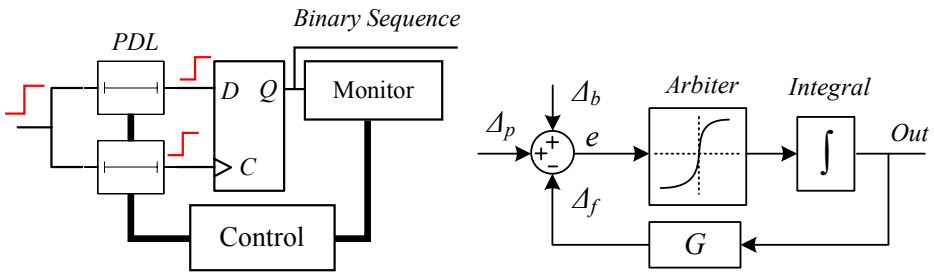


Figure A.1 : The TRNG system model.

A straightforward implementation of the monitoring unit can be realized by using

a counter. The counter value is incremented every time the flip-flop outputs '1' and is decremented whenever the flip-flop generates a '0'. This is analogous to performing a running sum over the sequence of output bits where zeros are replaced by '-1'. If zeros and ones are equally likely, the value of the counter will stay almost constant. A feedback signal is generated proportional to any deviation from this constant steady state value. The generated error signal is fed back to the signal-to-delay transducer, i.e., the PDL. The delay difference (Δ) is updated/corrected based on the feedback signal.

The described system is in effect a proportional-integral (PI) controller. The system is depicted in Figure A.1 (b). In this figure, Δ_b is the constant bias/skew in delays caused by the routing asymmetries. Δ_p is the delay difference induced by changes in environmental and operational conditions such as temperature and supply voltage, and/or delay difference imposed by active adversarial attacks. Δ_f is the correction feedback delay difference injected by the PDL based on the counter value. Equation A.1 expresses the total delay difference at the input of the flip-flop. G represents transformation carried out by the PDL from the counter binary value to an analog delay difference. The arbiter and integrator refer to the flip-flop and counter respectively. Therefore, the following relationship holds;

$$\Delta = \Delta_p + \Delta_b - \Delta_f. \tag{A.1}$$

An example PDL-based implementation of the TRNG system is shown in Figure A.2. The PDLs are depicted as gray triangles which provide the finest and most granular level of control over the delays. If the resulting delay difference from one PDL is equal to δ, the effective input/output delay of a PDL, $D(i)$, for the binary input i would be:

$$D(i) = i \times d_c + (1 - i) \times (d_c + \delta). \tag{A.2}$$

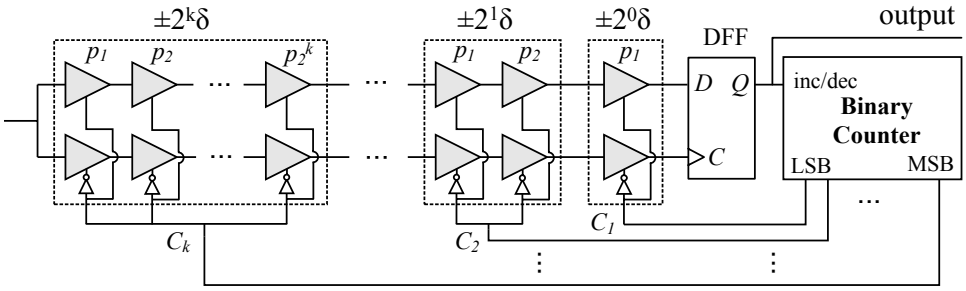


Figure A.2 : The TRNG system implementation with a PI controller on FPGA.

where d_c is a constant delay value. Each programmable delay block consists of two PDLs. The control input of top PDL inside each block is the complement of the bottom PDL control input in order to make a differential programmable delay structure. Based on Equation A.3, the differential delay is:

$$D_{diff}(i) = (1 - 2i) \times \delta = (-1)^i \delta, \qquad i = 0 \text{ or } 1. \tag{A.3}$$

In this example, the programmable delay blocks are packed in groups with sizes of multiples of two to efficiently generate any desirable delay difference using a binary control input. In other words, the first programmable delay block consists of two PDLs, the second one contains 4 PDLs, and so on. With this arrangement, the total incurred delay difference can be written as:

$$\Delta_f = G(\mathbf{C}) = \sum_{i=0}^{K} (-1)^{C_i} 2^i \delta, \tag{A.4}$$

where $C_i \in \mathbf{C}$ is the i^{th} counter bit with $i = 0$ being the least significant bit (LSB) and $i = K$ being the most significant bit (MSB), and $\mathbf{C}$ represents the counter value. δ is the smallest possible delay difference produced by one PDL.

Let us assume that in the beginning the counter is reset to zero. The resulting feedback delay difference is $\Delta_f = (2^{(K+1)} - 1) \times \delta$ according to Equation A.4. This large delay difference skews the output of flip-flop toward '1'. This keeps raising the counter value, lowering the delay difference (Δ). As Δ approaches zero, the flip-flop begins to output '0's more frequently and lowers the rate at which the counter value was previously increasing. At the steady state, the counter value will settle around a constant value with a slight oscillatory behavior. Any outside perturbation on delays will cause transient fluctuations in bit probabilities; however, the automatic adjustment mechanism brings the system back to the equilibrium state.

Counter	I^t	I^b	w
111	1111	0000	+4
110	0111	0000	+3
101	0011	0000	+2
100	0001	0000	+1
000	0000	0001	−1
001	0000	0011	−2
010	0000	0111	−3
011	0000	1111	−4

Figure A.3 : Decoding operation.

Although the performance of the system in Figure A.2 seems ideally flawless, a straightforward hardware implementation was not successful. This is because the design is based on the assumption that δs from PDLs are equal. However, due to manufacturing process variability, the δs slightly vary from one PDL to another. As a result, it is not feasible to generate any desirable delay difference, because the intended weights are not exactly multiples of two anymore. In particular, the input to the largest programmable delay block dominates the system's output behavior.

Instead, we took an alternative approach and used two sets of fine and coarse delay tuning blocks as shown in Figure A.4. With n fine tuning delay lines with a resolution of δ_{fn}, and m coarse tuning delay line with resolution of δ_{cs}, any delay difference in the range of $R = [n\delta_{fn} + m\delta_{cs}, -n\delta_{fn} - m\delta_{cs}]$ that satisfies Equation A.5

can be produced.

$$\Delta_f = w_{fn}\delta_{fn} + w_{cs}\delta_{cs} \tag{A.5}$$

where w_{fn} and w_{cs} are integer weights (or levels) such that $-n < w_{fn} < n$ and $-m < w_{cs} < m$. By carefully selecting n,m, δ_{fn}, and δ_{cs}, any delay difference with a resolution of δ_{fn} can be produced within the range R.

The system in Figure A.4 is designed such that the weights (or tuning levels) in Equation A.5 are a function of the difference in the total number of '1's at PDL inputs on the top and bottom paths;

$$w_{fn} = \sum_{i=1}^{n} I^t[i] - \sum_{i=1}^{n} I^b[i], \quad w_{cs} = \sum_{i=1}^{m} I^t[i] - \sum_{i=1}^{m} I^b[i] \tag{A.6}$$

where $I^t[i] \in \{0,1\}$ and $I^b[i] \in \{0,1\}$ are the input signals to PDLs as demonstrated in Figure A.4. Thus, decoder block in Figure A.4 needs to perform a mapping from the counter value to the number of '1's at PDL inputs. For example, if $n = 4$, the counter value of '111' corresponds to -4 and '000' corresponds to +4. Table A.3 shows an example of decoding operation and corresponding tuning weights for a 3-bit counter. The conversion from the counter value to the effective tuning weight is expressed by Equation A.7.

$$w_{fn} = (-1)^{C_K} \times \left(1 + \sum_{i=0}^{K-1} C_i 2^i\right), \quad K = \lfloor log_2 n \rfloor. \tag{A.7}$$

The fundamentals of the system's operation shown in Figure A.4 are the same as the system in Figure A.2 with the only difference lying in how the feedback signal is generated based on the counter states.

Notice that the controller type determines the response time to changes in delays as well as the error in the steady state response. Proportional integral (PI) controllers as opposed to proportional integral derivative (PID) controller due to the lack of

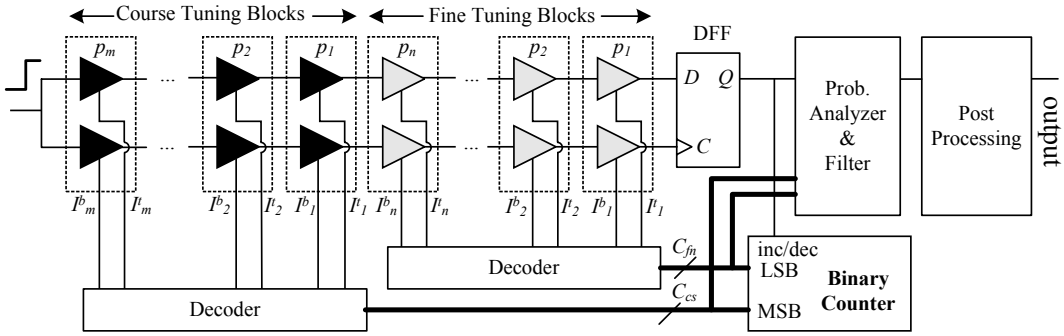


Figure A.4 : The complete TRNG system.

derivative function can make the system more stable in the steady state in the case of noisy data. This is because derivative action is more sensitive to higher-frequency terms in the inputs. Additionally, a PI-controlled system is less responsive to inputs (including noise) and so the system will be slower to respond to quick perturbations on the delays than a well-tuned PID system.

The following two observations are important from a security standpoint. First, in the steady state, the counter value oscillates around a constant center value (C_{center}). Let us define the oscillation amplitude as the peak-to-peak range of the oscillations, i.e. the maximum counter value minus the minimum counter value ($C_{max} - C_{min}$). The oscillation is not as periodic as one might think. It is rather a random walk around the center value. Each step in the random walk involves going from one counter value to a one lower or higher value:

$$\text{Step} : C_{current} \rightarrow C_{current} \pm 1$$

The probability of each step (move) is a function of the current location. Intuitively

the probability of going outside the range is almost zero:

$$\begin{aligned} \text{Prob}\{C_{max} \rightarrow C_{max} + 1\} &\simeq 0 \\ \text{Prob}\{C_{min} \rightarrow C_{min} - 1\} &\simeq 0 \end{aligned} \tag{A.8}$$

Also assuming a smooth monotonically increasing probability curve as shown in Figure A.5 for the flip-flop, the farther the current counter value is from the center (C_{center}), the lower the probability of moving farther away from the center:

$$\begin{aligned} Prob\{C_i \rightarrow C_i + 1\} < Prob\{C_j \rightarrow C_j + 1\} \text{ for } C_j < C_i \\ Prob\{C_i \rightarrow C_i - 1\} < Prob\{C_j \rightarrow C_j - 1\} \text{ for } C_j < C_i \end{aligned} \tag{A.9}$$

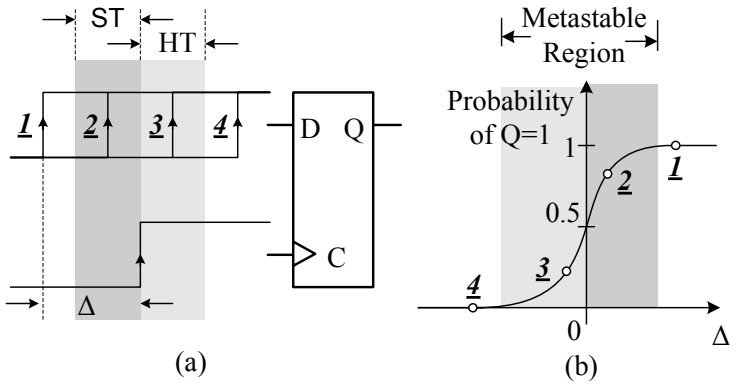


Figure A.5 : (a) Flip-flop operation under four sampling scenarios, (b) probability of output being equal to '1' as a function of the input signals delay difference (Δ). The numbers on the probability plot correspond to each signal arrival scenario.

Each generated output bit corresponds to a counter value. The probability of the output being to '1' is a function of the feedback counter value. The maximum counter value almost always results in a '0' output, since a '0' value decrements the counter value. Based on Equation A.8, transition $C_{max} \rightarrow C_{max} + 1$ is unlikely, thus $r(C_{max})$

can almost never be '1'. The following deductions can be explained similarly:

$$Prob\{r(C_{center}) = 1\} \simeq 0.5$$
$$Prob\{r(C_{min}) = 1\} \simeq 1$$
$$Prob\{r(C_{max}) = 1\} \simeq 0 \tag{A.10}$$

In other words, during the random walk only those steps that pass close at the center point will result in high entropy and non-deterministic responses. A smaller error in the steady state response means oscillations happen closer to center of the probability transition curve which in turn leads to higher randomness in generated output bits.

In addition, it is desired that the system responds as quickly as possible to external perturbations since the during the recovery time the TRNG generates output bits with highly skewed probabilities.

A.2 Experimental results

In this section, we present the LUT-based PDL delay measurement evaluations and TRNG hardware implementation results obtained from Xilinx Virtex 5 LX50T FPGA.

Before moving onto the TRNG system performance evaluation, we shall first discuss the results of our investigation on the maximum achievable resolution of the PDLs. We set up a highly accurate delay measurement system similar to the delay characterization systems presented in Section 4.1 of Chapter 4.

The circuit under test consists of four PDLs each implemented by a single 6-input LUT. The delay measurement circuit as shown in Figure A.6 consists of three flip-flops: launch, sample, and capture flip-flops. At each rising edge of the clock, the launch flip-flop successively sends a low-to-high and high-to-low signal through the

PDLs. At the falling edge of the clock, the output from the last PDL is sampled by the sample flip-flop. At the last PDL's output, the sampled signal is compared with the steady state signal. If the signal has already arrived at the sample flip-flop when the sampling takes place, then these two values will be the same; Otherwise they take on different values. In case of inconsistency in sampled and actual values, XOR output becomes high, which indicates a timing error. The capture flip-flop holds the XOR output for one clock cycle.

To measure the absolute delays, the clock frequency is swept from a low frequency to a high target frequency and the rate at which timing errors occur are monitored and recorded. Timing errors start to emerge when the clock half period (T/2) approaches the delay of the circuit under test. Around this point, the timing error rate begins to increase from 0% and reaches 100%. The center of this transition curve marks the point where the clock half period (T/2) is equal to the effective delay of the circuit under test.

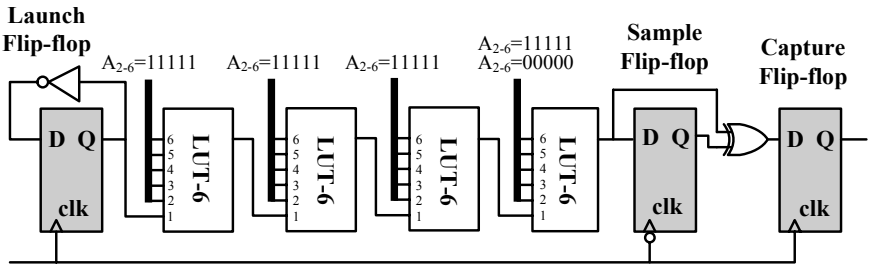


Figure A.6 : The delay measurement circuit. The circuit under test consists of four LUTs each implementing a PDL.

To measure the delay difference incurred by the LUT-based PDL, the measurement is performed twice using different inputs. In the first round of measurement, the inputs to the four PDLs are fixed to $A_{2-6} = 11111$. In the second measurement the inputs to the last PDL are changed to $A_{2-6} = 00000$. In our setup, a 32×32 array of

the circuit shown on Figure A.6 is implemented on a Xilinx Virtex 5 LX110 FPGA, and the delay from our setup is measured under the two input settings. The clock frequency is swept linearly from 8MHz to 20MHz using a desktop function generator and this frequency is shifted up by 34 times inside the FPGA using the built-in PLL.

To evaluate the performance of the TRNG system, we implement the system shown in Figure A.4 using 32 coarse and fine programmable delay lines ($n = m = 32$). A 12-bit counter performs the running sum operation on the output generated bits. The first six (LSB) bits control the finely tunable PDLs, and the next six (MSB) bits control the coarsely tunable PDLs. Both fine and coarse PDLs are implemented by using one LUT as shown in Figure A.7. As illustrated in Figure A.7, to implement the fine PDL, the LUT inputs A_3 to A_6 are fixed to zero and the only input that controls the delay is A_2. For the coarse PDL, all of the LUT inputs are tied and controlled together.

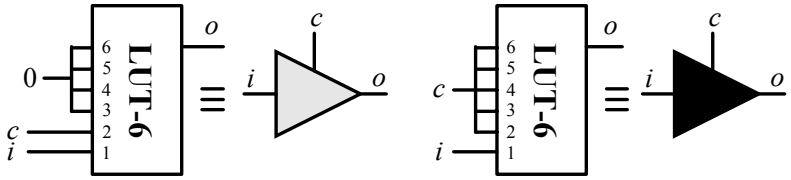


Figure A.7 : Coarse and fine PDLs implemented by a single 6-input LUT.

In the first experiment, we only examine the forward system, which consists of the PDLs, the flip-flop, and the decoders. The tuning weights/levels are swept from the minimum to maximum, and the probability of the flip-flop producing a '1' output is measured at each level. This probability is measured by repeating each experiment over 100 times and counting the number of times the flip-flop outputs a '1'. Since

$n = m = 32$, both the fine and coarse tuning levels can go from -32 to 32. Recall that the tuning level represents the difference in the total number of ones at PDL inputs on the top path minus those on the bottom path (see Equation A.6). As can be observed from Figure A.8, increasing both the coarse and fine tuning levels increase the probability of output being equal to '1'. The non-smoothness of the probability curve is due to variability in the manufacturing process which creates local non-monotonicity. With these observations, we expect the feedback system

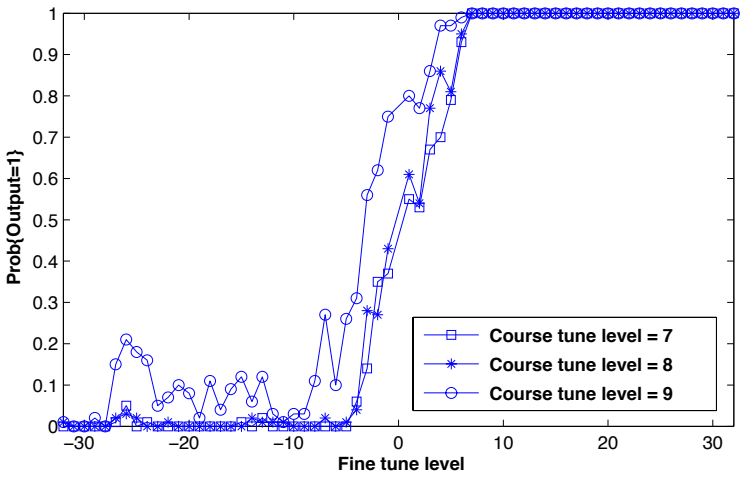


Figure A.8 : The probability of flip-flip generating a '1' output as a function of the fine and coarse tuning levels.

behavior to stabilize somewhere close to the center of the transition point. Next, we close the feedback loop and initialize the operation. At the beginning, the counter is loaded with all '1's (which results in a decimal value of 2^{12}-1 = 4095). Figure A.9 shows the counter value as the operation progresses. The x-axis is the number of clock cycles. Once the operation starts, the counter value keeps decreasing until it reaches the value of approximately 700 after about 3,400 clock cycles. From this point further, the counter value reaches a steady state with a slight oscillatory behavior around a

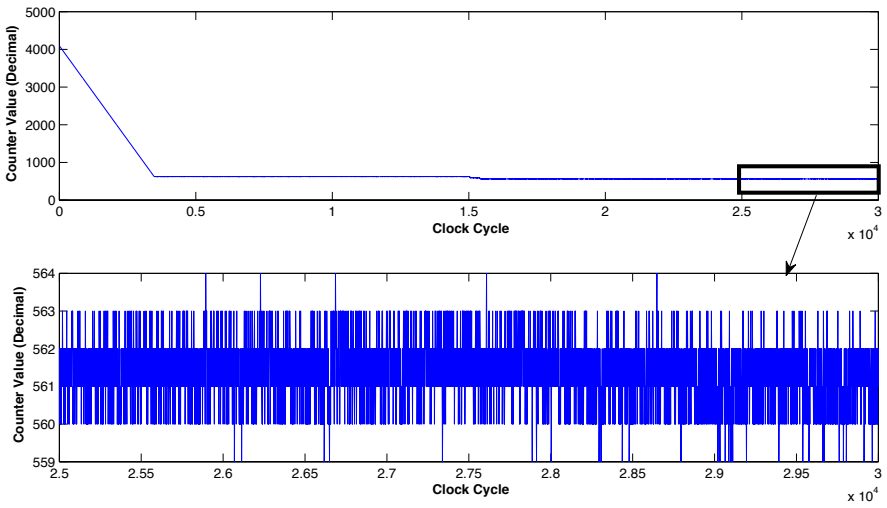


Figure A.9 : The transient counter value (decimal) versus the clock cycles.

constant value. A close-up of the steady state behavior is depicted in the lower plot of Figure A.9. The close-up zooms into the segment between 25,000 to 30,000 clock cycles. As can be observed in the steady state, the counter value oscillates between 559 and 564.

Next, we investigate the frequencies at which counter values appear in the steady state. In this experiment, we collect 1,000,000 counter values in the steady state and plot the histogram of the observed values as shown in the middle plot (b) in Figure A.10. The normalized histogram suggests that the counter holds the value of 561 more than 40% of the time. Next, it is critical to investigate the probabilities associated with each courter value. In other words, we would like to know for the given counter values – which produce a feedback input to the TRNG core – the probability of the flip-flop output being equal to '1'. The top plot (a) in Figure A.10 presents this result. It is interesting to see that most of the counter values produce highly skewed probabilities. Among these counter values, 561 leads to a '1' output

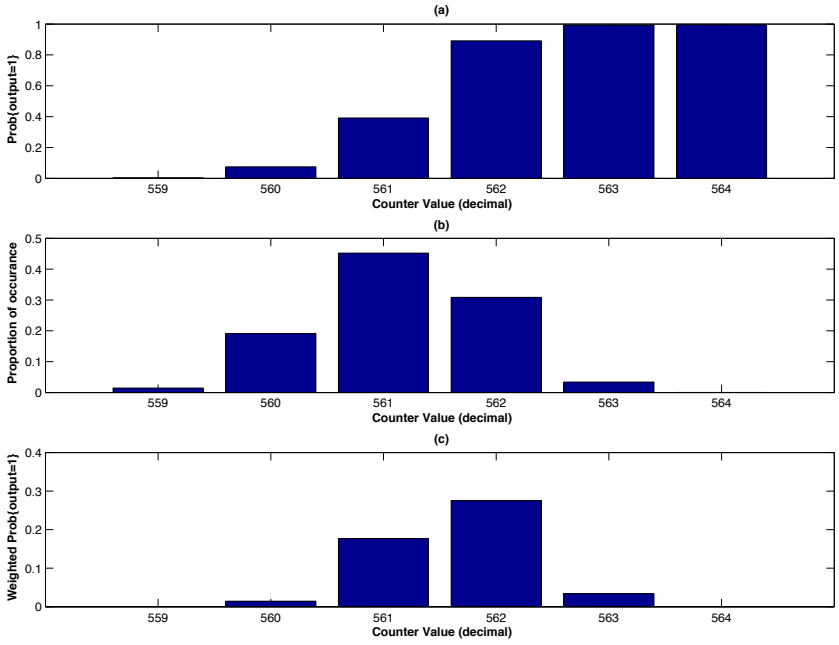


Figure A.10 : Distribution of the steady state counter values and associated bit probabilities.

slightly more than 40% of the time. We define a metric which is the multiplication of the counter values' frequency of occurrence with the probability of output being equal to one for each counter value. This metric represents the contribution of each counter value to the total number of '1' in the output sequence. The metric values are shown in bottom plot (c) in Figure A.10.

To remove the bias in the output sequence in a systematic way as well as to eliminate predictable patterns, we propose a filtering mechanism based on the steady state counter values. The filter unit analyzes the output bit probabilities for each counter value within a window of specific size and flags the counter values that lead to outputs bits with skewed probabilities. Next, it filters out the output bits associated with the flagged counter values. For example, in our implementation, the filter only

allows output bits associated with the counter value of 561 to pass through. As a result, the bit-rate is lowered to almost half of the original bit-rate. However, the output bits may still suffer from bias in the bit probabilities. Therefore, a post-processing unit after the filter unit is used to remove any localized biases from the bitstream. In our implementation, we use a von Neumann corrector to perform the post-processing task. The results of the NIST randomness test from running on megabytes of data is shown in Table A.1. The comprehensive test results are available online at http://www.ruf.rice.edu/ mm7/trng/.

Table A.1 includes the results of the NIST statistical test suite on megabytes of collected data after counter-based filtering and von Neumann correction are performed on the TRNG output bitstream. Due to the large bias in the probabilities, most of the randomness failed when the test was run on the output bitstream before the filtering and correction were carried out.

Finally, according to the ISE Synthesis report, the propagation delay through the TRNG core is equal to 61.06ns which achieves a bit-rate of 16Mbit/sec. The bit-rate drops to 1/8 of the original bit-rate (to 2Mbit/sec) after filtering and von Neumann correction. The TRNG core consumes 128 LUTs that are packed into 16 Virtex 5 CLBs. Note that in practice multiple TRNG cores can run in parallel to offer a higher bit-rate.

Table A.1 : NIST Statistical Test Suite results.

Statistical Test	Block/Template length	Lowest success ratio
Frequency	-	100%
Frequency within blocks	128	100%
Cumulative sums	-	100%
Runs	-	100%
Longest run within blocks	-	100%
Binary rank	-	100%
FFT	-	100%
Non-overlapping templates	9	90%
Overlapping templates	9	100%
Maurer's universal test	7	100%
Approximate entropy	10	100%
Random excursions	-	100%
Serial	16	100%
Linear complexity	500	90%

Appendix B

Plots

In this appendix, a comprehensive list of measurement results and plots for each FPGA is shown. The Figures B.1 through B.12 show the transition in the number of '1's in responses as the tuning level is swept from 40 to -10. At each fixed tuning level, 64,000 challenges are fed to the PUF and 64,000 responses are collected. The best tuning level is the one for which the PUF responses are half zeros and half ones. Each line ine the following figures corresponds to each of the PUF output bits. Since each PUF on each FPGA produces 16 response bits, there are 16 lines on each subplot. There are 9 subplots in each plot. Each subplot corresponds to the measurement taken under a different operating condition. The center subplot refers to the normal supply voltage and room temperature.

The Figures B.13 to B.24 show the robustness of the responses to different subset of challenges. Each challenge to the arbiter PUF creates a delay difference (Δ) at the input of the arbiter (flip-flop). The Δs produced by all challenges in the challenge space form a Gaussian distribution. If half of the responses are one and half are zero, then this distribution has a mean of zero. The distribution is split by the arbiter decision border (line). Those challenges that create a Δ that is larger that e, result in a '1' response and a zero response otherwise. e is basically the arbiter bias remained after tuning. We partition the Δ distribution and the corresponding challenge space into 20 sets of equal size. The Δs close to the decision border are more sensitive to environmental condition fluctuations, and those farther apart from

the decision border (i.e. $-\Delta$ - e $- \gg 0$) are less affected by such fluctuations. The following figures quantify these effects for each of the twenty challenge sub-sets. The x-axis in each subplot refers to the challenge partition (bin) number. Each partition contains $64000/20 = 3200$ challenges. The y-axis shows the stability of the corresponding responses, where '1' means no errors in the responses and '0' means completely erroneous responses. The error is measured by comparing the responses from eight corner cases to the response at the normal operating condition (room temperature and nominal supply voltage). Therefore, each subplot contains eight lines for each corner case. As it can be observed the challenges in bins that are closer to the decision border produce responses with larger error rates. There are 16 subplot in each figure where each correspond to a PUF output response bit.

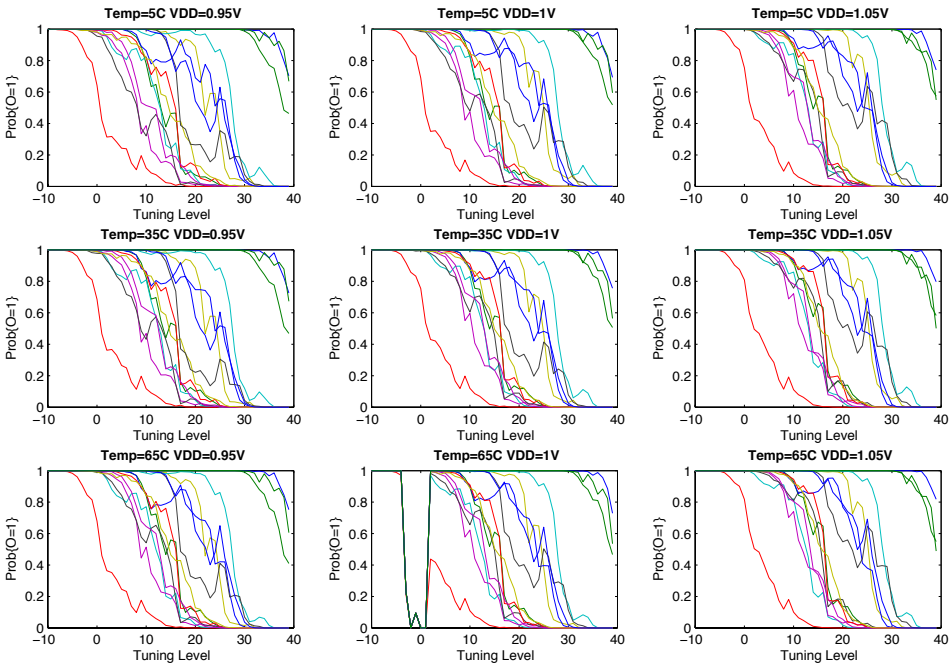


Figure B.1 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 6.

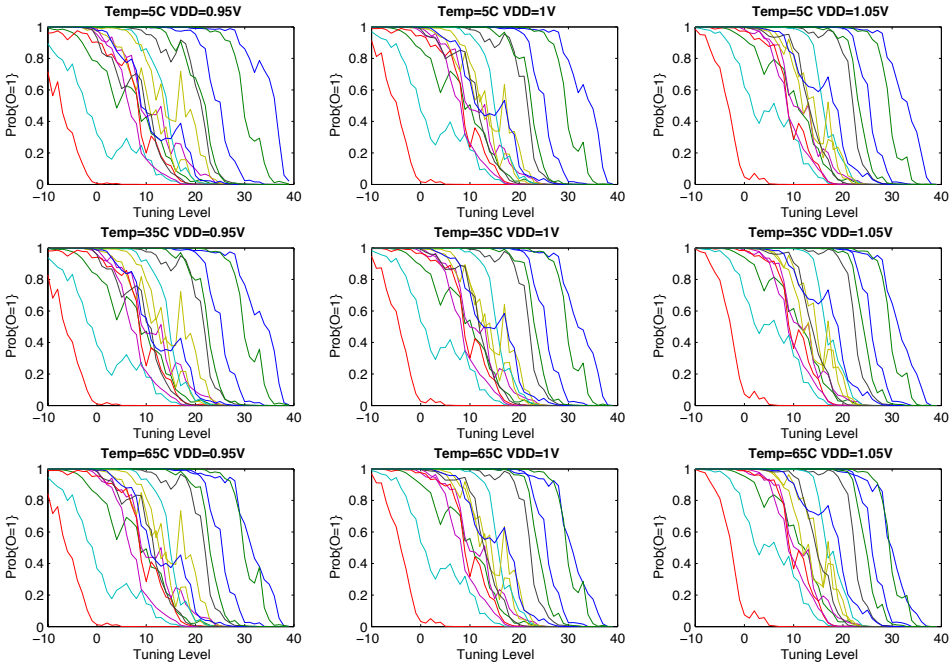


Figure B.2 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 7.

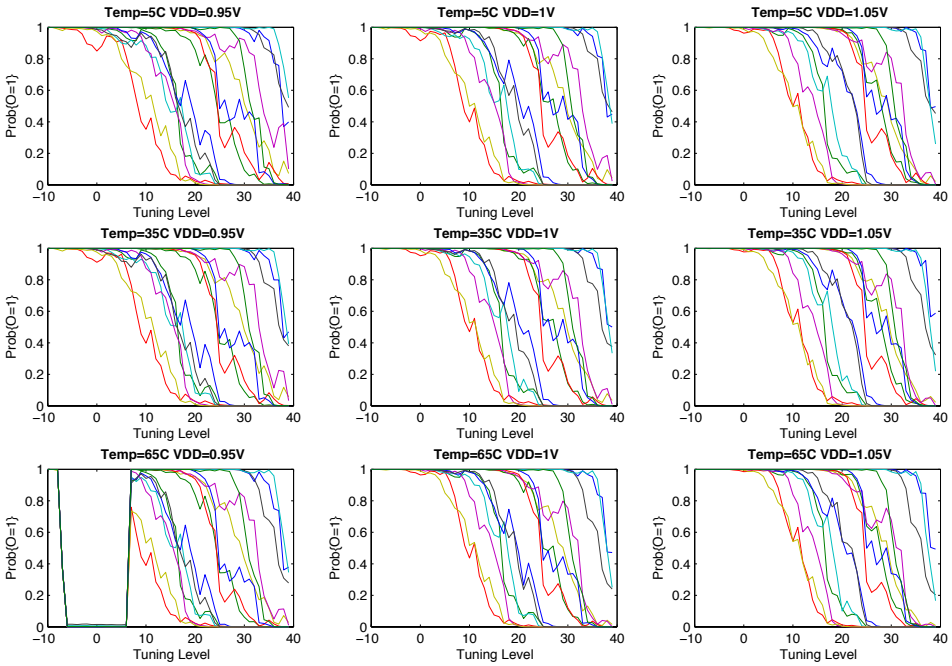


Figure B.3 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 8.

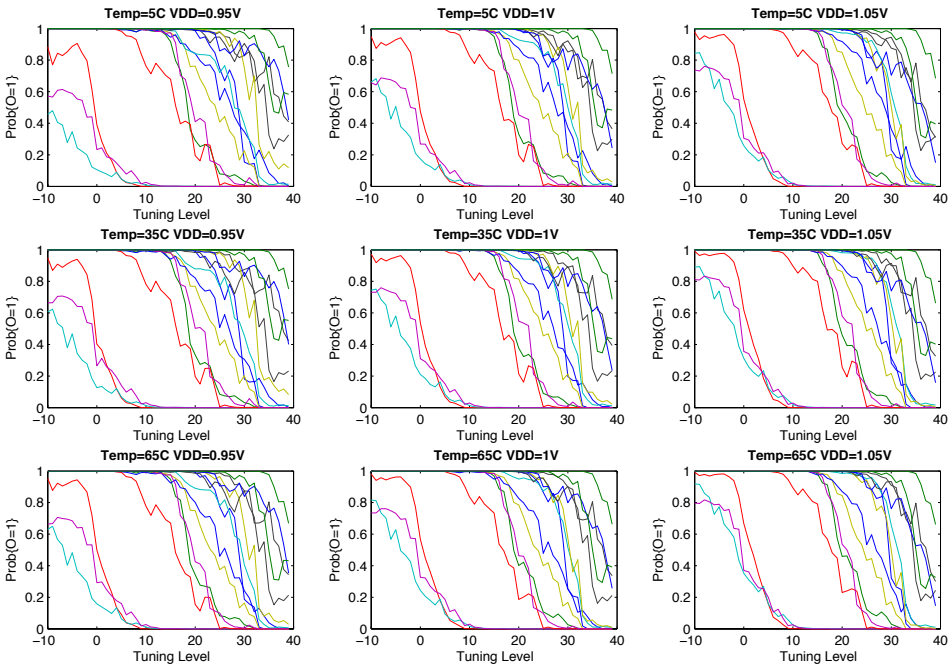


Figure B.4 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 9.

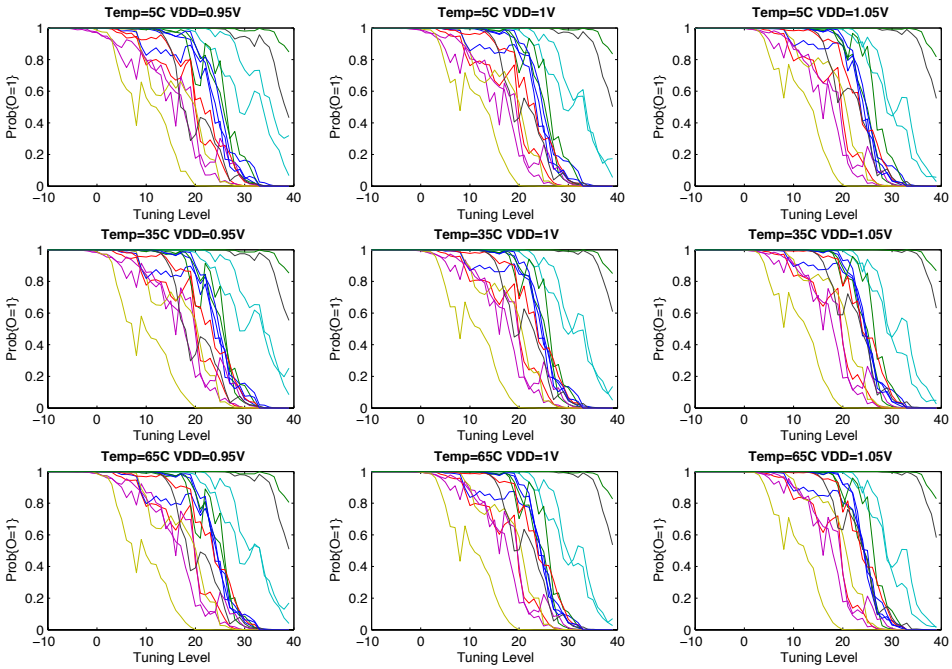


Figure B.5 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 10.

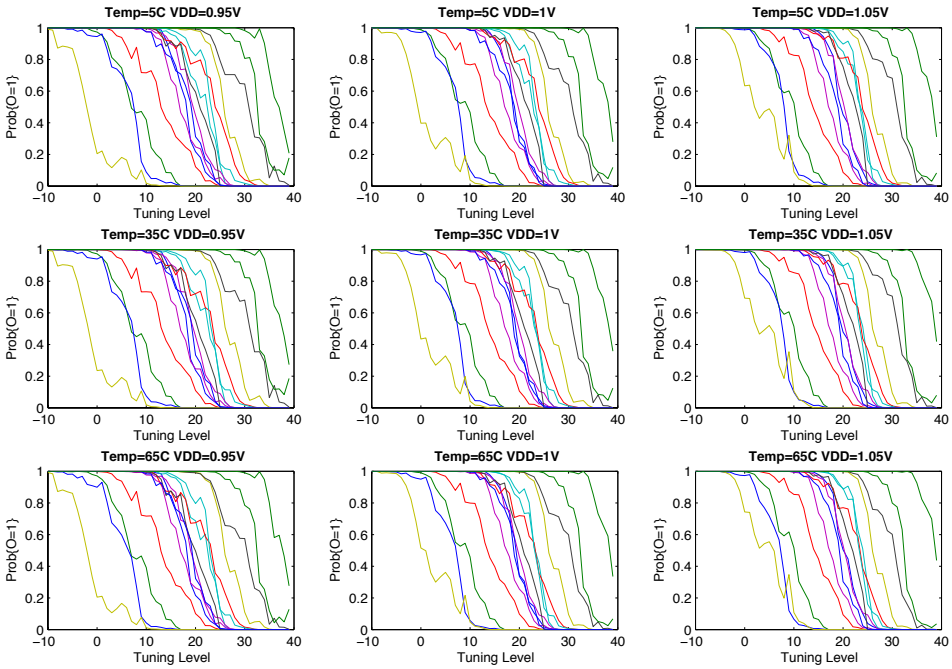


Figure B.6 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 11.

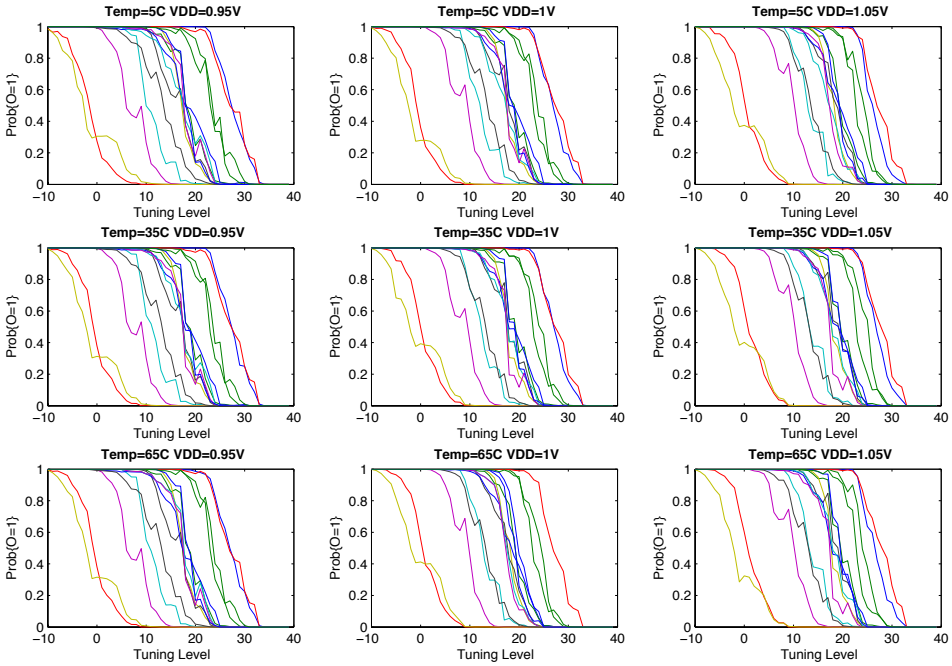


Figure B.7 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 12.

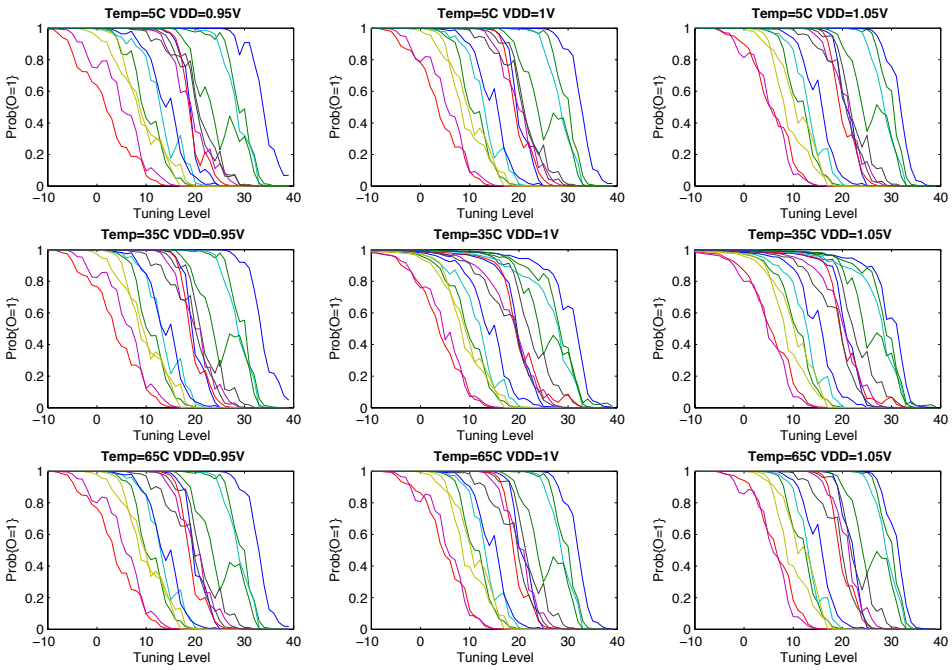


Figure B.8 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 13.

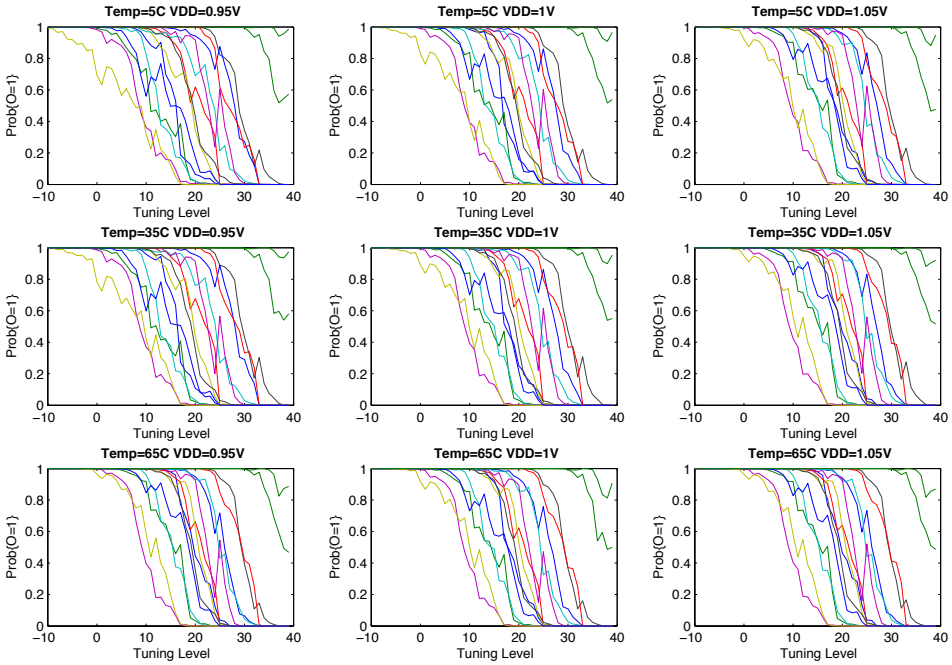


Figure B.9 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 14.

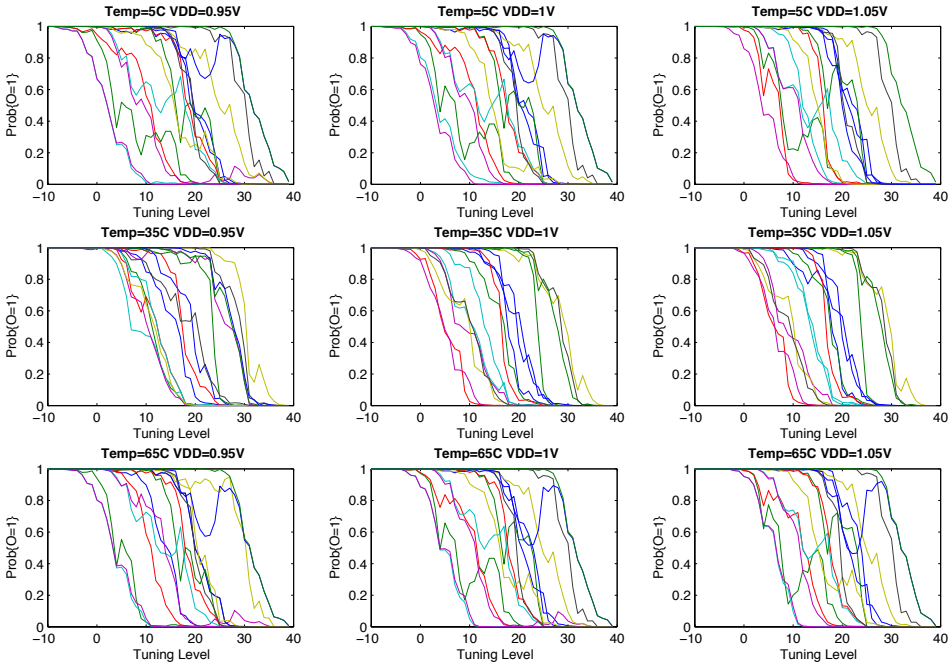


Figure B.10 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 15.

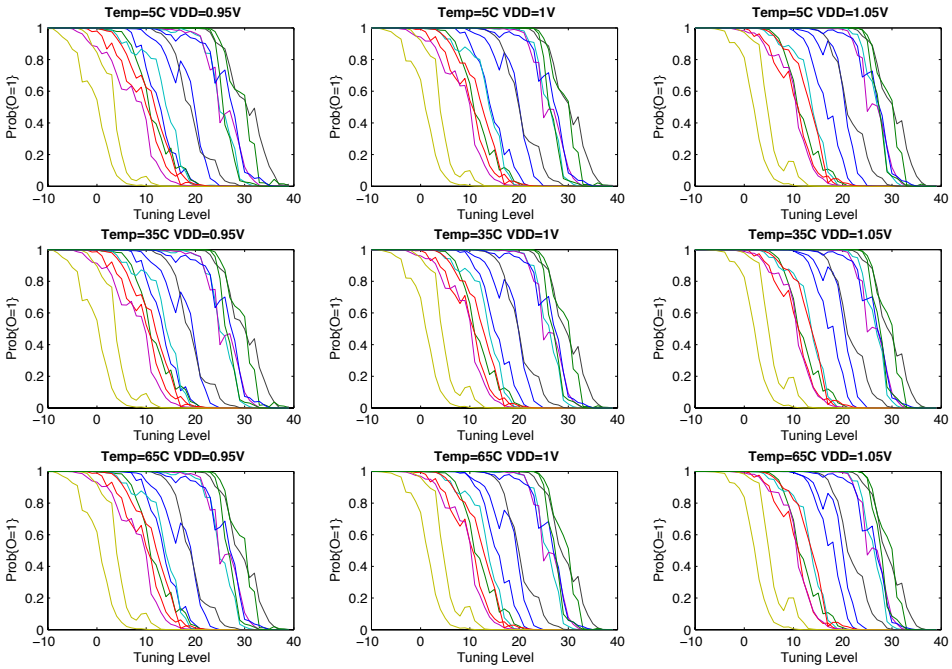


Figure B.11 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 16.

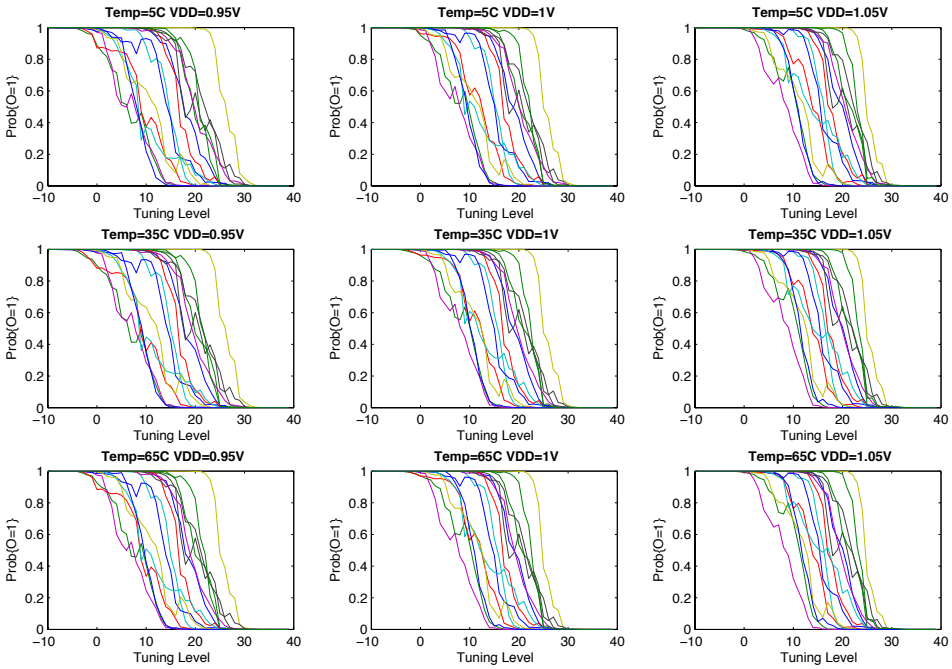


Figure B.12 : Number of '1's in responses (normalized) as a function of tuning level for the PUF on FPGA 17.

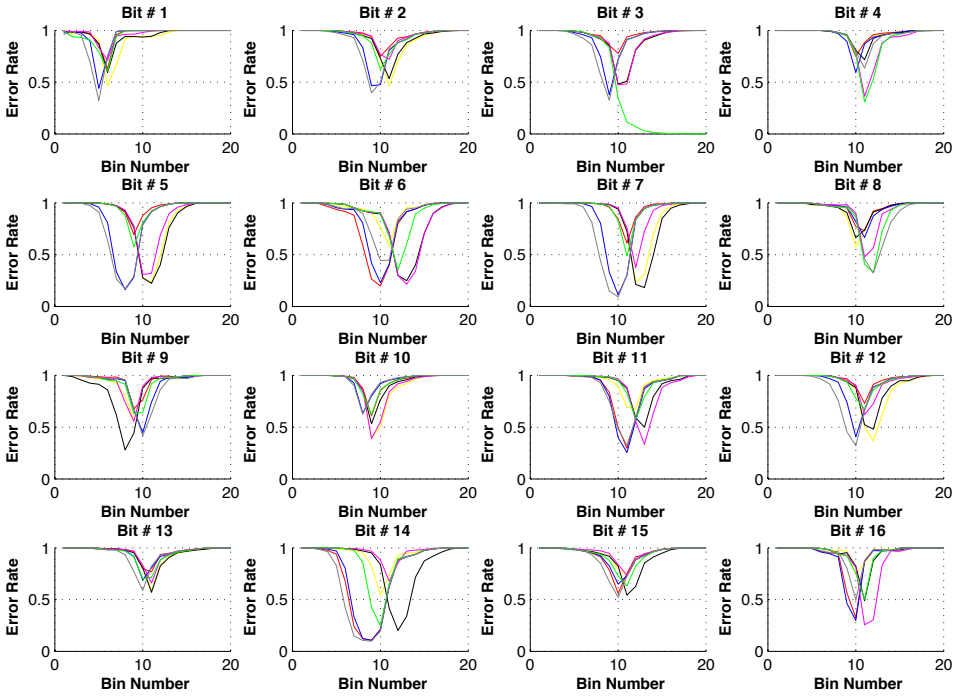


Figure B.13 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 6.

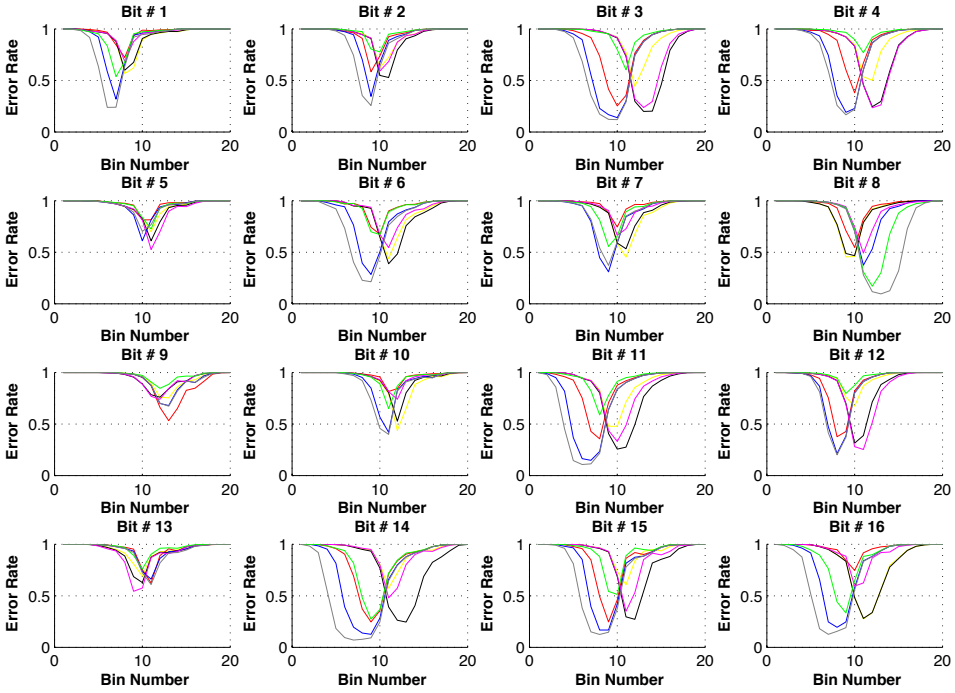


Figure B.14 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 7.

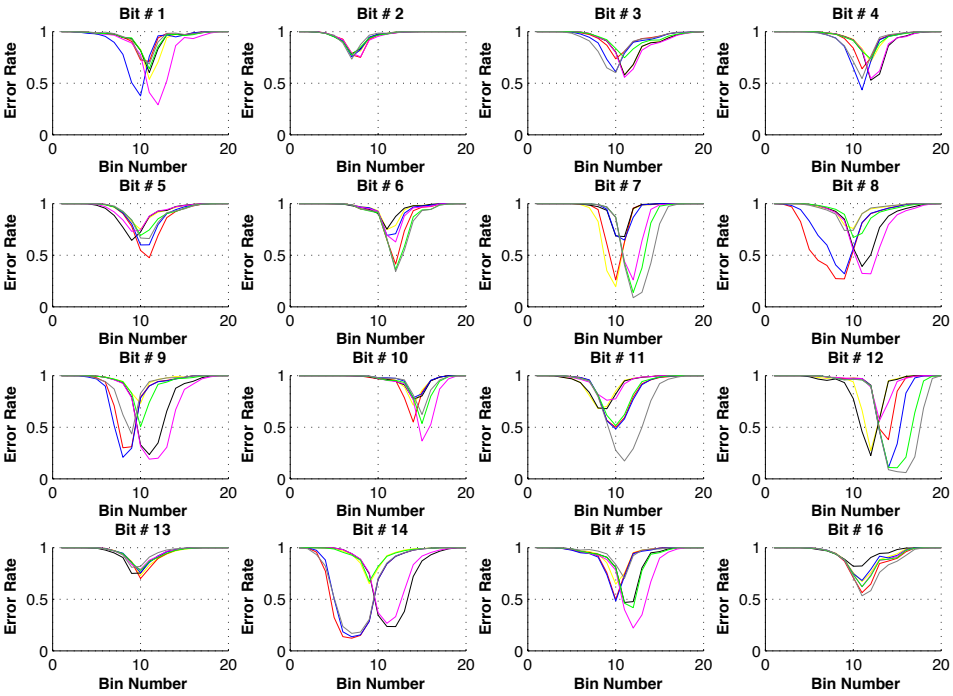


Figure B.15 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 8.

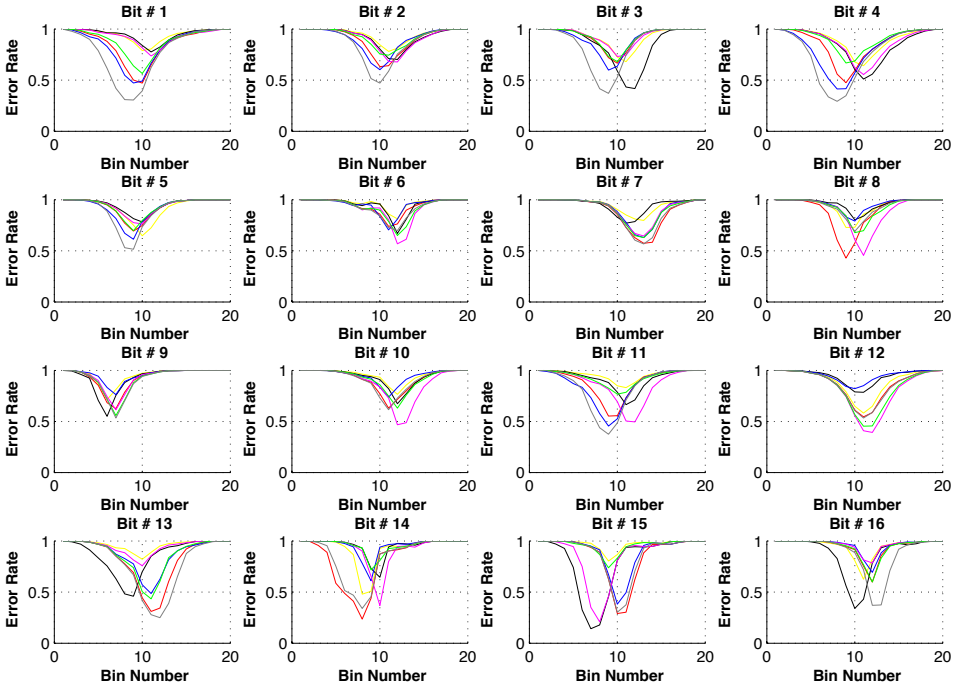


Figure B.16 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 9.

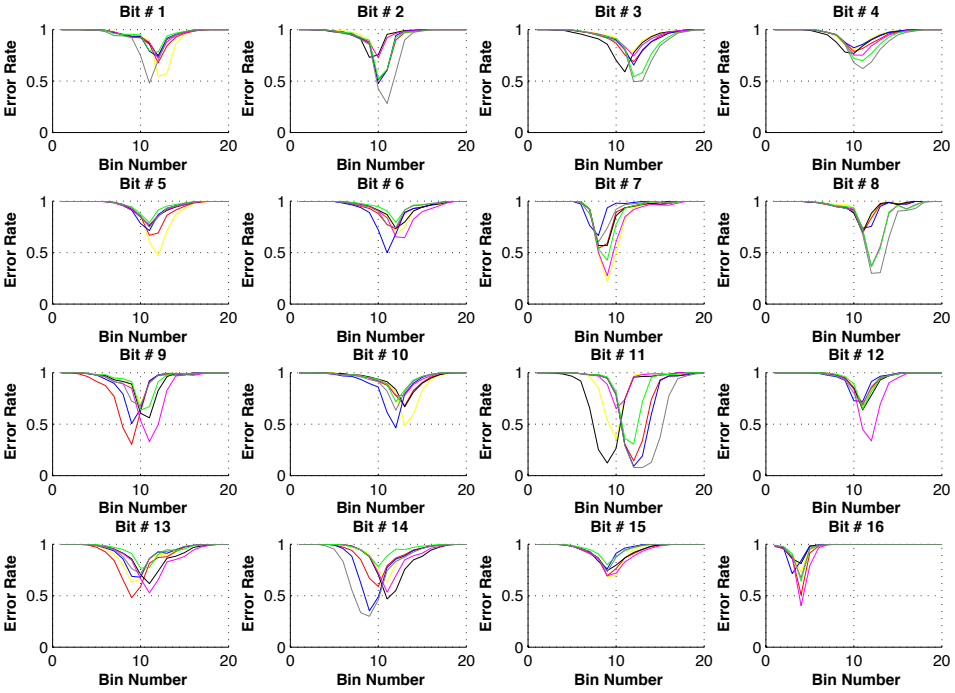


Figure B.17 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 10.

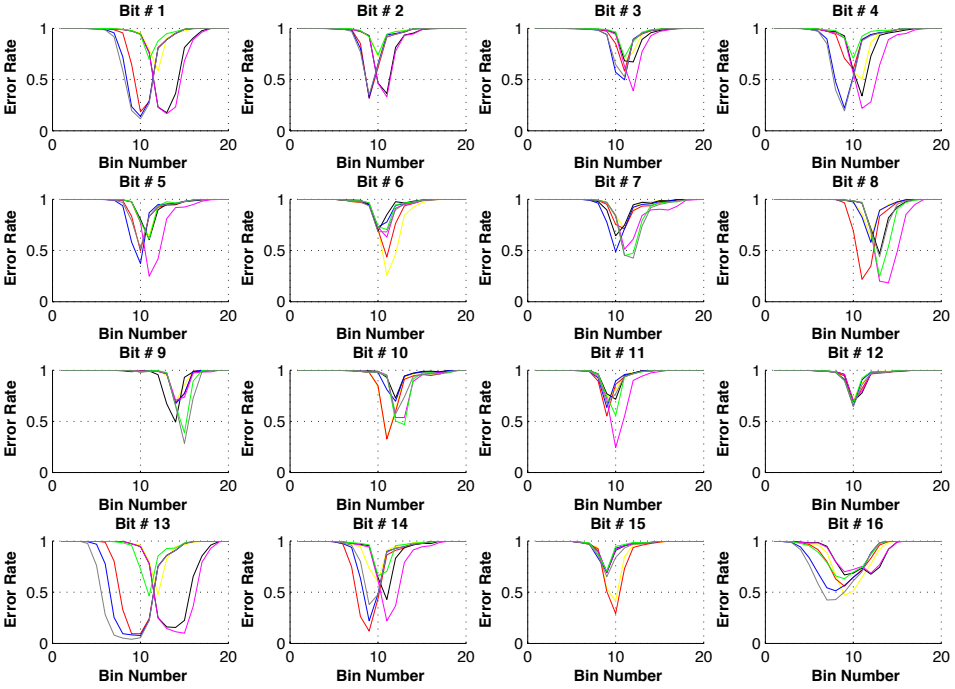


Figure B.18 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 11.

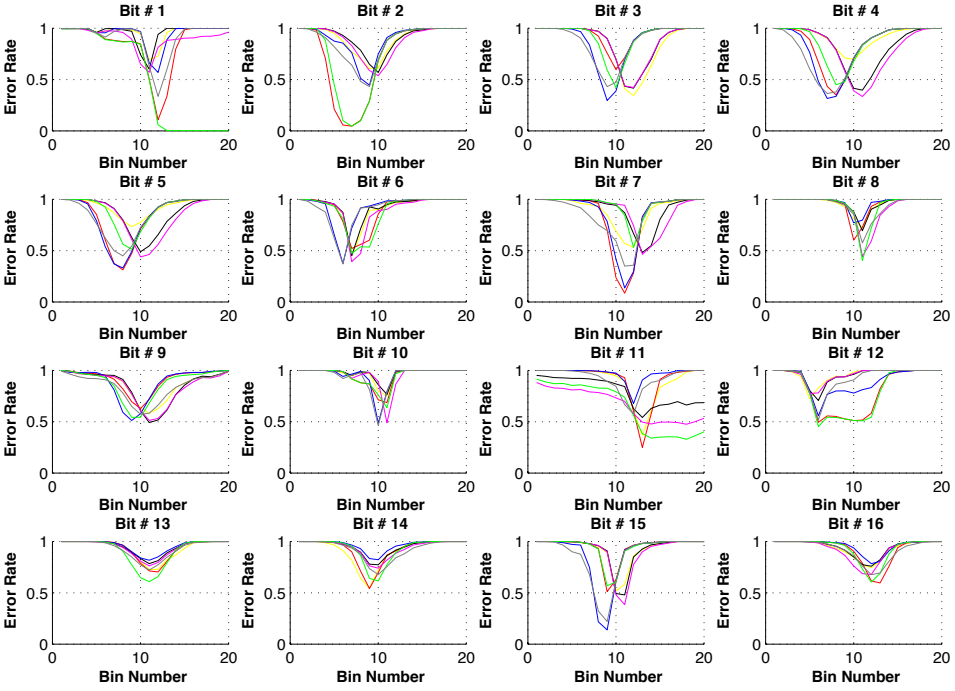


Figure B.19 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 12.

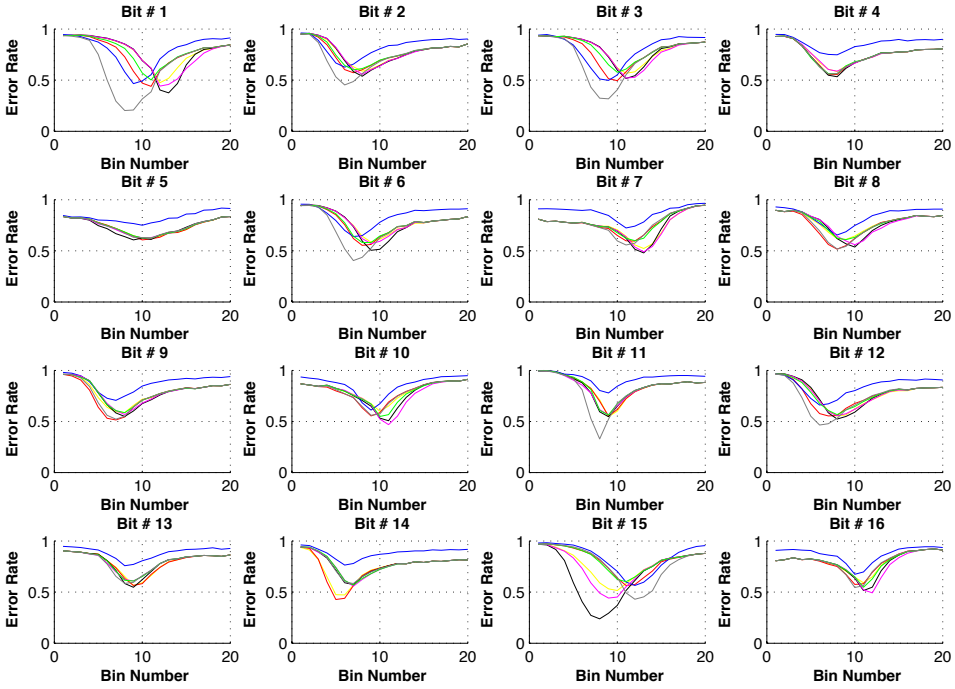


Figure B.20 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 13.

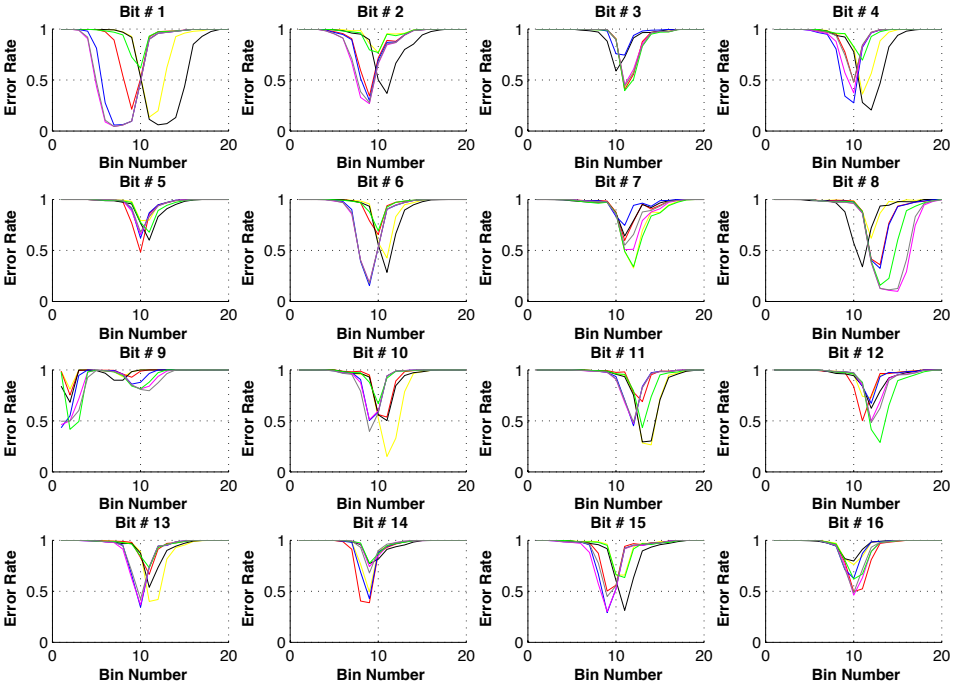


Figure B.21 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 14.

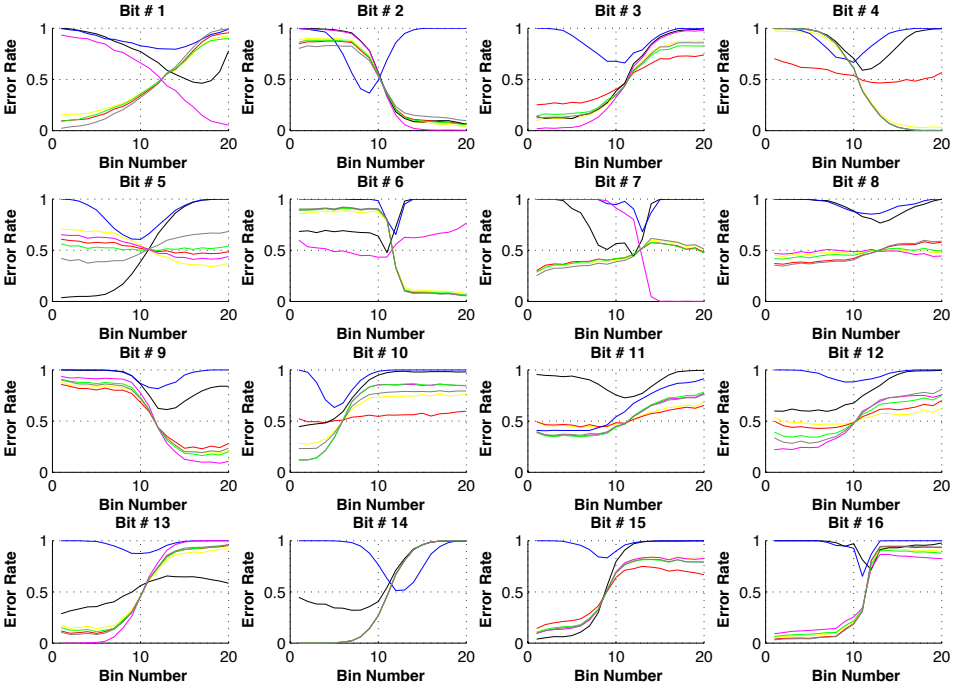


Figure B.22 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 15.

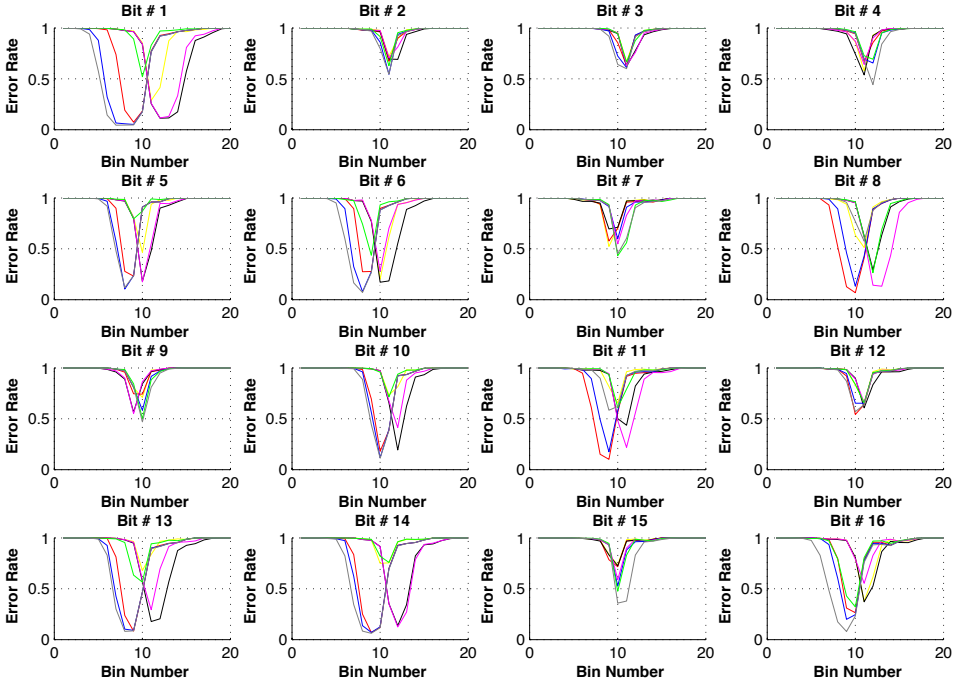


Figure B.23 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 16.

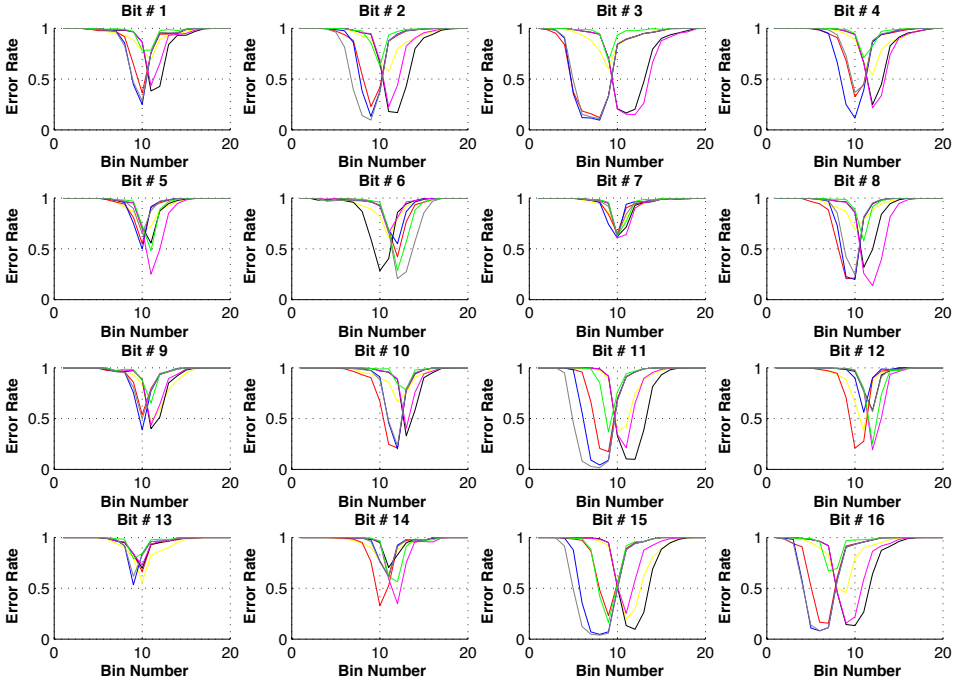


Figure B.24 : Response stability measured across different challenge partitions with reference to eight operating condition corner cases for FPGA 17.

Bibliography

[1] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *CCS*, 2002, pp. 148–160.

[2] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *International Test Conference (ITC)*, 2008, pp. 1–10.

[3] R. McEvoy, F. Crowe, C. Murphy, and W. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on.* IEEE, 2006, pp. 6–pp.

[4] G. Suh, C. O'Donnell, and S. Devadas, "AEGIS: a Single-Chip secure processor," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 570–580, 2007.

[5] J. Guajardo, S. Kumar, G. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *CHES*, 2007, pp. 63–80.

[6] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference (DAC)*, 2007, pp. 9–14.

[7] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," *Cryptographic Hardware and Embedded Systems–CHES 2011*, pp. 17–32, 2011.

[8] M. Majzoobi, "Techniques for design and implementation of physically unclonable functions," Master's thesis, Rice University, 2009.

[9] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 1, pp. 1–33, 2009.

[10] M. Majzoobi, G. Ghiaasi, F. Koushanfar, and S. Nassif, "Ultra-low power current-based PUF," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 2071–2074.

[11] M. Majzoobi, F. Koushanfar, and M. Potkonjak, FPGA-*oriented Security*. Springer, 2011.

[12] M. Majzoobi and F. Koushanfar, "Time-bounded authentication of FPGAs," in *Under Revision for IEEE Trans. on Information Forensics and Security (TIFS)*, 2011.

[13] M. Majzoobi, A. E. Nably, and F. Koushanfar, "FPGA time-bounded authentication," in *Information Hiding Conference (IH)*, 2010, pp. 1–15.

[14] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, "Rapid FPGA delay characterization using clock synthesis and sparse sampling," in *International Test Conference (ITC)*, 2010, p. in press.

[15] E. Dyer, M. Majzoobi, and F. Koushanfar, "Hybrid modeling of non-stationary process variations," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, 2011, pp. 194 –199.

[16] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," 2010, pp. 1 – 6.

[17] S. Morozov, A. Maiti, and P. Schaumont, *An Analysis of Delay Based* PUF *Implementations on* FPGA. Springer, 2010, pp. 382–387.

[18] S. Drimer and S. J. Murdoch, "Keep your enemies close: distance bounding against smartcard relay attacks," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 2007, pp. 7:1–7:16.

[19] K. B. Rasmussen and S. Čapkun, "Realization of rf distance bounding," in *Proceedings of the 19th USENIX conference on Security*, 2010, pp. 25–25.

[20] M. Majzoobi, M. Rostami, F. Koushanfar, D. Wallach, and S. Devadas, "Slender PUF Protocol: A lightweight, robust, and secure authentication by substring matching," in *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, may 2012, pp. 33 –44.

[21] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2011, pp. 128 –133.

[22] 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 with identification and authentication applications," in *IEEE VLSI Circuits Symposium*, New-York, June 2004.

[23] C. Yin and G. Qu, "LISA: maximizing RO PUF's secret extraction," in *Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 100–105.

[24] G. Qu and C.-E. Yin, "Temperature-aware cooperative ring oscillator PUF," in *Hardware-Oriented Security and Trust (HOST)*, 2009, pp. 36–42.

[25] A. Maiti and P. Schaumont, "Improved ring oscillator PUF: An FPGA-friendly secure primitive," *Journal of Cryptology*, pp. 1–23, 2010.

[26] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in *Hardware-Oriented Security and Trust (HOST)*, june 2010, pp. 94 –99.

[27] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, 2002.

[28] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *ACSAC*, 2002, pp. 149–160.

[29] D. Holcomb, W. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, September 2009.

[30] U. Rührmair, "SIMPL system: on a public key variant of physical unclonable function," *Cryptology ePrint Archive*, 2009.

[31] S. Kumar, J. Guajardo, R. Maes, G. Schrijen, and P. Tuyls, "The butterfly PUF protecting IP on every FPGA," in *HOST*, 2008, pp. 67–70.

[32] S. Ying, J. Holleman, and B. Otis, "A digital 1.6 pj/bit chip identification circuit using process variations," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 69 –77, jan. 2008.

[33] K. Lofstrom, W. Daasch, and D. Taylor, "IC identification circuit using device mismatch," 2000, pp. 372 –373.

[34] J. S. J. Wong, P. Sedcole, and P. Y. K. Cheung, "Self-measurement of combinatorial circuit delays in FPGAs," *ACM TRETS*, vol. 2, no. 2, pp. 1–22, 2009.

[35] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[36] D. Rai and J. Lach, "Performance of delay-based trojan detection techniques under parameter variations," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2009, pp. 58–65.

[37] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz, "Serecon: A secure dynamic partial reconfiguration controller," *IEEE Computer Society Annual Symposium on VLSI*, vol. 0, pp. 292–297, 2008.

[38] J. B. Webb and J. B. Webb, "Methods for securing the integrity of FPGA configurations," Tech. Rep., 2006.

[39] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, and R. Vaslin, "Reconfigurable hardware for high-security/ high-performance embedded systems: The safes perspective," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 144 –155, 2008.

[40] B. Gassend, "Physical random functions," S.M. Thesis, MIT, 2003.

[41] U. Ruhrmair, "SIMPL system: on a public key variant of physical unclonable function," *Cryptology ePrint Archive*, 2009.

[42] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, Massachusetts Institute of Technology, may 2004.

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

[44] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 237–249.

[45] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *ICCAD*, 2008, pp. 670–673.

[46] C. Bösche, J. G. A. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2008, pp. 181–197.

[47] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2009, pp. 332–347.

[48] M.-D. M. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design and Test of Computers*, vol. 27, pp. 48–65, 2010.

[49] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *EUROCRYPT*, 2004, pp. 523–540.

[50] B. Sunar, W. Martin, and D. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *Computers, IEEE Transactions on*, vol. 56, no. 1, pp. 109–119, 2007.

[51] D. Schellekens, B. Preneel, and I. Verbauwhede, "FPGA vendor agnostic true random number generator," in *Field Programmable Logic and Applications (FPL)*, 2006, pp. 1 –6.

[52] C. W. Odonnell, G. E. Suh, and S. Devadas, "PUF-based random number generation," in *In MIT CSAIL CSG Technical Memo 481*, 2004.

[53] J. von Neumann, "Various techniques used in connection with random digits," *von Neumann Collected Works*, vol. 5, pp. 768–770, 1963.

[54] B. Barak, R. Shaltiel, and E. Tromer, "True random number generators secure in a changing environment," in *Cryptographic Hardware and Embedded Systems workshop (CHES)*. Springer-Verlag, 2003, pp. 166–180.

[55] B. Jun and P. Kocher, "The Intel random number generator," in *CRYPTOGRAPHY RESEARCH, INC.*, 1999.

[56] B. Sunar, *Cryptographic engineering*. Springer, 2009, ch. True Random Number Generators for Cryptography.

[57] L. Daihyun, J. Lee, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on VLSI Systems*, vol. 13, no. 10, pp. 1200 – 1205, 2005.

[58] U. Rührmair, Q. Chen, M. Stutzmann, P. Lugli, U. Schlichtmann, and G. Csaba, "Towards electrical, integrated implementations of simpl systems," in *Workshop*

in Information Security Theory and Practice (WISTP), 2010, pp. 277–292.

[59] K. Eguro, "Sirc: An extensible reconfigurable computing communication api," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 135–138.

[60] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," in *Information Hiding*, 2009, pp. 206–220.

[61] J. Note and E. Rannaud, "From the bitstream to the netlist," in *FPGA*, 2008, pp. 264–274.

[62] M. Baldi, F. Chiaraluce, N. Boujnah, and R. Garello, "On the autocorrelation properties of truncated maximum-length sequences and their effect on the power spectrum," *Signal Processing, IEEE Transactions on*, vol. 58, 2010.

[63] C. K. Koc, Ed., *Cryptographic Engineering*, 1st ed. Springer, Dec. 2008.