April 2018

Applications Of Physical Unclonable Functions on ASICS and FPGAs

Mohammad Usmani
University of Massachusetts Amherst

APPLICATIONS OF PHYSICAL UNCLONABLE FUNCTIONS ON ASICS AND FPGAS

A Thesis Presented

by

MOHAMMAD A USMANI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2018

Electrical and Computer Engineering

APPLICATIONS OF PHYSICAL UNCLONABLE FUNCTIONS ON ASICS AND FPGAS

A Thesis Presented

by

MOHAMMAD A USMANI

Approved as to style and content by:

Daniel Holcomb, Chair

Russell Tessier, Member

Jay Taneja, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGMENTS

Thanks to Professor Daniel Holcomb for his guidance on this thesis and all the support from the beginning of time in this new country. Also thanks to Professor Russell Tessier for his guidance on many sections of this thesis.

ABSTRACT

APPLICATIONS OF PHYSICAL UNCLONABLE FUNCTIONS ON ASICS AND FPGAS

FEBRUARY 2018

MOHAMMAD A USMANI

B.Tech., ALIGARH MUSLIM UNIVERSITY

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Daniel Holcomb

With the ever-increasing demand of security in embedded systems and wireless sensor networks, we require integrating security primitives for authentication in these devices. One such primitive is known as a Physically Unclonable Function. This entity can be used to provide security at a low cost, as the key or digital signature can be generated by dedicating a small part of the silicon die to these primitives which produces a fingerprint unique to each device. This fingerprint produced by a PUF is called its response. The response of PUFs depends upon the process variation that occurs during the manufacturing process. In embedded systems and especially wireless sensor networks, there is a need to secure the data the collected from the sensors.

To tackle this problem, we propose the use of SRAM-based PUFs to detect the temperature of the system. This is done by taking the PUF response to generate temperature based keys. The key would act as proofs of temperature of the system.

In SRAM PUFs, it is experimentally determined that at varying temperatures there is a shift in the response of the cells from zero to one and vice-versa. This variation can be exploited to generate random but repeatable keys at different temperatures.

To evaluate our approach, we first analyze the key metrics of a PUF, namely, reliability and uniqueness. In order to test the idea of using the PUF as a temperature based key generator, we collect data from a total of ten SRAM chips at fixed temperatures steps. We first calculate the reliability, which is related to bit error rate, an important parameter with respect to error correction, at various temperatures to verify the stability of the responses. We then identify the temperature of the system by using a temperature sensor and then encode the key offset by PUF response at that temperature using BCH codes. This key-temperature pair can then be used to establish secure communication between the nodes. Thus, this scheme helps in establishing secure keys as the generation has an extra variable to produce confusion.

We developed a novel PUF for Xilinx FPGAs and evaluated its quality metrics. It is very compact and has high uniqueness and reliability. We also implement 2 different PUF configurations to allow per-device selection of best PUFs to reduce the area and power required for key-generation. We also evaluate the temperature response of this PUF and show improvement in the response by using per-device selection.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . . **iv**

ABSTRACT . . . . . **v**

LIST OF TABLES . . . . . **x**

LIST OF FIGURES . . . . . **xi**

CHAPTER

1. INTRODUCTION **1**

1.1 Trends 1
1.2 Thesis overview 2
1.3 Thesis outline 4

2. BACKGROUND **5**

2.1 Arbiter PUF 6
2.2 Ring Oscillator PUF 6
2.3 Butterfly PUF 7
2.4 Sensor based PUF 8
2.5 Bistable Ring PUF 9
2.6 Anderson PUF 10
2.7 Challenges associated with PUFs 12

3. STATIC RANDOM ACCESS MEMORY BASED PHYSICAL UNCLONABLE FUNCTION **13**

3.1 SRAM design 13
3.2 SRAM PUF operation 14
3.3 SRAM chip architecture 16
3.4 Experimental validation 17

3.4.1 Uniqueness 18

3.4.2 Reliability . . . 19

3.5 Results and analysis . . . 19

4. PUF BASED KEYS AS PROOFS OF TEMPERATURE . . . 22

4.1 Main objective . . . 22
4.2 Data Collection . . . 23

4.2.1 Data remanence . . . 23
4.2.2 Data-collection setup . . . 25
4.2.3 Temperature response of SRAM . . . 26
4.2.4 Testing on advanced technology SRAM . . . 27
4.2.5 Sensitivity analysis . . . 29

4.3 Per temperature key enrollment and generation system . . . 33

4.3.1 Key enrollment . . . 33
4.3.2 Key generation . . . 34
4.3.3 Complete system . . . 35

4.4 Conclusion . . . 36

5. ALTERNATE IMPLEMENTATION OF ANDERSON'S PUF ON XILINX FPGA . . . 37

5.1 Motivation . . . 37

5.1.1 Discrete stages for tune-ability . . . 38
5.1.2 Design resource requirement . . . 38

5.2 The SLICEL PUF implementation and characterization . . . 40

5.2.1 PUF Hamming weight tuning . . . 41
5.2.2 PUF reliability & uniqueness . . . 45
5.2.3 Spatial Autocorrelation of PUF Location BERs . . . 46

5.3 Per-Device selection of PUFs . . . 48

5.3.1 Correlation of between-configuration PUF responses . . . 52
5.3.2 Design flow for per-device PUF configuration . . . 54
5.3.3 Temperature dependence of PUF response . . . 55

6. PUF BASED KEY GENERATION ON FPGAS . . . 58

6.1 Cost of Error correction . . . 60
6.2 A statistical model for PUF error correction . . . 60

6.2.1 Two parameter model 61
6.2.2 PUF model generation 62

6.3 Results 64

7. CONCLUSION 67

BIBLIOGRAPHY 68

LIST OF TABLES

Table **Page**

3.1 SRAM chips under evaluation 17

3.2 PUF reliability and uniqueness for 3 different SRAM chips evaluated using 128-bit PUF blocks 20

4.1 Between temperature average Hamming distances observed on 1 chip 28

4.2 Between temperature average Hamming distances observed on 1 chip after Majority voting 29

4.3 Between temperature average Hamming distances observed on an 160nm 23LC1024 chip 31

4.4 Between temperature average Hamming distances observed on 160nm 23LC1024 chip after Majority voting 31

4.5 Between temperature average Hamming distances observed on a 90nm CY7C185 chip 32

4.6 Between temperature average Hamming distances observed on a 90nm CY7C185 chip after Majority voting 32

5.1 Mean Hamming distance comparisons from 10 chips 49

5.2 Within class Hamming distance of PUFs (4250-bit) under different configurations at different temperatures 57

6.1 Resource requirements for 128-bit key generation 66

LIST OF FIGURES

Figure **Page**

1.1 Proposed system 3

1.2 Figure showing the restriction of resources on an FPGA chip for reconfigurable PUF selection and making the rest of the design independent 3

2.1 Arbiter PUF [11] 5

2.2 Basic Ring Oscillator PUF 7

2.3 Ring Oscillator PUF using multiple ring oscillators and a counter [41] 7

2.4 Butterfly PUF [24] 8

2.5 Sensor based PUF [33] 8

2.6 Basic bistable ring 9

2.7 Bistable Ring PUF[5] 9

2.8 Anderson's PUF on Xilinx FPGA[3] 11

3.1 SRAM chip circuitry [38] 15

3.2 SRAM cell [15] 16

3.3 Figure showing effect of process variations and noise on SRAM cell 16

3.4 AS6C6264 SRAM chip details[2] 17

3.5 Between and within class Hamming distances for 128 bit PUFs from AS6C6264 chips 20

3.6 Between and within class Hamming distances for 128 bit PUFs from 23LC1024 chips 21

3.7 Between and within class Hamming distances for 128 bit PUFs from CY7C185 chips 21

4.1 Data remanence duration after power down at different temperatures 24

4.2 System for data collection at different temperatures 25

4.3 BER of PUF instances placed on AS6C6264 chip 1 across different temperatures 27

4.4 BER of PUF instances placed on AS6C6264 chip 2 across different temperatures 28

4.5 BER of PUF instances placed on chip 23LC1024 across different temperatures 30

4.6 BER of PUF instances placed on chip CY7C185 across different temperatures 30

4.7 Figure showing the sensitivity of Within class Hamming distance with respect to temperatures for different technology SRAMs. 33

4.8 Per-temperature one time key enrollment 34

4.9 Per-temperature key generation 35

4.10 Complete system 36

5.1 Figure showing the effect of elongating the carry chain on SLICEM based PUF's response. 38

5.2 Figure showing the effect of adding Multiplexer stages in the chain. The glitch get filtered out, thereby narrowing it down and reducing the probability of setting the Flip-Flop. 39

5.3 Figure showing variation of Hamming weight with change in delay difference between two paths for a carry chain of length 2. 39

5.4 The alternate implementation of Anderson's PUF [3] implemented on a Virtex 7 architecture. The two LUTs are separated vertically by three Multiplexer stages and feed their corresponding flip-flops to generate a toggle signal for the select lines of their corresponding Muxes. 42

5.5 The timing waveforms shown govern the operation of glitch generation and glitch filtering in SLICEL PUF. 43

5.6 Histograms of between-class and within-class Hamming distances of 128-bit PUFs. Within-class distances compare two measurements from the same 128-bit PUF instance. Between-class distances compare (in b) two different 128-bit PUFs on the same chip, or (in c) compare 128-bit PUFs that occupy the same locations on different chips. All measurements were made at room temperature of approximately 24^o and at the nominal supply voltage. 47

5.7 Two possible PUF configurations within the cell 50

5.8 Figure shows the BER of PUF instances placed at different locations on a chip. Unreliable instances are scattered and not concentrated in a particular area of the chip and uncorrelated across configurations. 51

5.9 Figure showing the distribution of a 1 response of the PUF across 10 chips. A high probability closer to 0 or 100 percent implies that PUFs are highly reliable. Broken line indicates points where the value is zero. 52

5.10 When the same logic slices are configured in two different ways (see Fig. 5.7) on the same chips: (a) their BERs are uncorrelated; and (b) the fractional Hamming distance between responses from each configuration is 38.02-bits (29.71%). 53

5.11 Effect of temperature on reliability of PUF(4250-bit) before best PUF configurations selection 55

5.12 Effect of temperature on reliability of PUF(4250-bit) after best PUF configuration selection 56

6.1 Block diagram showing the one time enrollment process for key generation 59

6.2 Block diagram showing the one time enrollment process for key generation 59

6.3 Fitted model vs actual data obtained at $\lambda_1 = 0.0711$ and $\lambda_2 = 0.1834$ for configuration 1 63

6.4 Key failure distribution with different number of error correcting BCH code for Configuration 1 64

6.5 Key failure distribution with different number of error correcting BCH code for Configuration 2 65

6.6 Key failure distribution with different number of error correcting BCH code for best configurations 65

CHAPTER 1

INTRODUCTION

1.1 Trends

Today the Internet of Things (IoT) and wireless sensor networks(WSNs) are used for an increasingly large number of applications which require security. Security measures are needed to prevent malicious access to the system or the network to prevent tampering of data and functionality. One such security measure can be implemented by using an encryption scheme. To implement an encryption scheme, each sensor node or device needs to have its own key. PUF can be used to generate secure cryptographic keys [30]. PUF-based keys are unique to each device as they stem from the process variations inherent in the silicon at the time of fabrication. The work proposed in this thesis develops a PUF based secure temperature sensor using the power-up state of SRAM cells. The idea for a secondary temperature sensor builds on the idea of a temperature-based virtual proof [34]. This temperature sensor can then be used in a key exchange scenario between two nodes since the power-up response of the SRAMs varies with temperature. The first node will report its temperature and send over a message with the key generated from the SRAM PUF at that temperature. The other node will already know the key at that specific temperature and will be able to decrypt the message. We will also explore the implementation of this scheme on FPGAs. We will develop a new type of PUF specific to the FPGAs. The following are the main areas of work:

- We have tested the SRAM PUF on an 8Kx8 SRAM chip AS6C6264 fabricated in 0.35μm technology and generate results for a temperature range of 0-55 degree celsius with a fixed step size.

- We then generate the plots for within temperature and between temperature Hamming distances to test how finely we can distinguish the temperatures.

- We have tested the scheme using advanced technology SRAM cells fabricated in 160 nm and 90 nm.

- We implemented a new type of PUF on a FPGA platform using design derived from [3]. The new design is implemented on SLICE-L cells. This will help us in restricting a block on FPGA dedicated to encryption and key generation.

- We propose and evaluate a per-device PUF configuration selection scheme on FPGA to save the cost of error correction and key generation and also improve temperature response.

1.2 Thesis overview

PUFs are primitives used in applications that demand high security. These are efficient to implement and require a small area overhead. In this thesis, we propose two different works done on PUFs.

First is a secure temperature based key generation scheme using the SRAM Physically Unclonable Function. Applications requiring highly secure key generation can leverage the change in response of the SRAM PUF with temperature to develop a tamper-proof communication system. In this thesis, we will focus on the development of a system for a secure key generation scheme based on temperature.

Fig. 1.1 shows the basic system. It consists of master and slave nodes communicating over an insecure link. The ambient temperature on the slave node can be verified based on the key generated for the encryption of its message. The reported

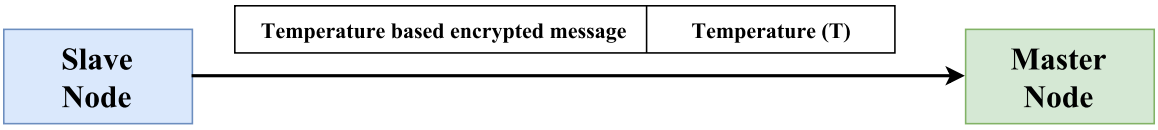


Figure 1.1: Proposed system

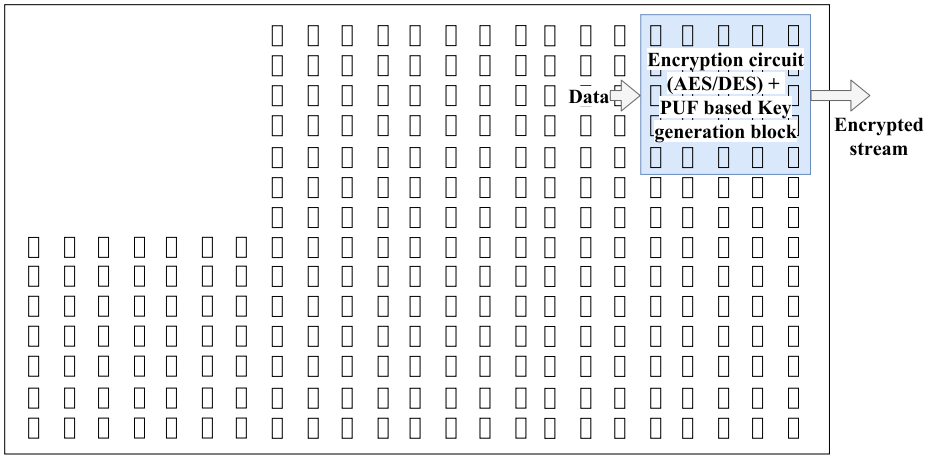


Figure 1.2: Figure showing the restriction of resources on an FPGA chip for reconfigurable PUF selection and making the rest of the design independent

temperature from the slave will be used to decrypt the message. If the decrypted message is invalid, we will know there is something wrong with the node, and it is trying to fake its temperature.

We will study the sensitivity of the SRAM PUF response to temperature to determine the minimum distinguishable change in temperature. The effects of temperature on the behavior of SRAM cells will be taken into account during the development of the system.

The second work proposed in this thesis is a novel implementation of Anderson's PUF implemented on Xilinx FPGAs. This implementation can be instantiated anywhere on the chip, unlike the original design which had specific design requirements. One advantage of this implementation is that we can restrict the block for encryption anywhere on the chip as shown in Fig. 1.2. This PUF can be instantiated in two different configurations at each location. We explore the advantage of selecting, on a

per-device basis, the best of the two PUF configuration for a reduction in overall bit error rate. We discuss the CAD flow for the per-device configuration and also present the area savings achieved using this approach. Lastly, we present the temperature response of the PUF showing its robustness across a wide range of temperatures and improvement in BER across temperatures gained by using the proposed approach of per-device configuration of PUFs.

1.3 Thesis outline

Chapter 2 reviews different types of PUFs that have been developed and implemented in ASICs and FPGAs. Chapter 3 covers the design of the SRAM PUF used in this work including detailed analysis about the uniqueness and reliability of SRAM PUFs is covered. Chapter 4 focuses on the core idea behind the use of PUF as a temperature sensor and generating temperature based keys. The advantages of using SRAM along with a temperature sensor for key establishment are discussed.

A novel PUF implementation on Xilinx FPGAs is discussed in chapter 5. We also discuss a per-device PUF selection scheme to improve the reliability of the PUFs and gain savings in the area.

CHAPTER 2

BACKGROUND

Physical Unclonable Functions (PUF) are primitives that produce unique chip specific signatures dynamically by exploiting the process variations inherent in the silicon during fabrication. PUFs can be classified into two types, namely strong PUF, and weak PUFs [35]. The strong PUFs can generate multiple random but repeatable responses by accepting challenges as input and mapping each challenge to a corresponding response in a way that is unique to each challenge. Weak PUFs, on the other hand, generate a single response. Both the types of PUFs have been implemented in ASICs and FPGAs. In this chapter, background of the various different types of PUFs is presented, more specifically the Arbiter PUF [11], the Ring Oscillator PUF [7], the Butterfly PUF [24], and the sensor based PUF [33, 43]. A comparison of various PUFs have been discussed in [21].

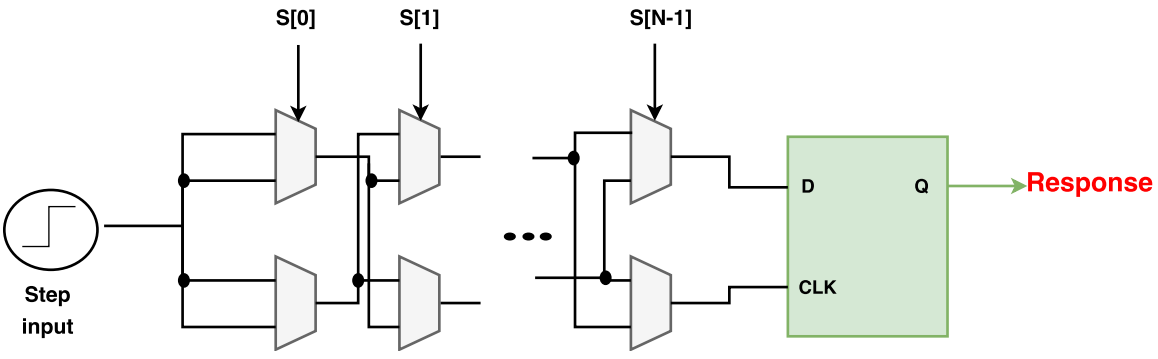


Figure 2.1: Arbiter PUF [11]

2.1 Arbiter PUF

The arbiter PUF[11, 26, 39, 4] is based on the delay of a parallel chain of multiple stages of multiplexers which feed a Flip-Flop. This PUF exploits the difference in the delay of the multiplexer paths. Figure 2.1 shows an arbiter PUF, with two parallel N-stage multiplexer chains feeding a flip-flop. A step signal is applied to the input and the step propagates through the paths of the multiplexer stages. The N-bit (S[0],...,S[N-1]) challenge is fed to the select line input of the multiplexers. Depending on the value of S[i] the signal will propagate through the upper path or the lower through stage i of the multiplexer chain. One path goes to the clock input and other to data input. If the step signal to data input reaches first, then a 1 is latched in the Flip-Flop otherwise, if the step signal reaches the clock input first, a 0 is latched. The arbiter-based PUF brings low resource overhead, but its structure makes it hard to map the multiplexer on matched paths on FPGA. This PUF is an example of strong PUF as there are various possible challenges that can be fed by the user to get different responses.

2.2 Ring Oscillator PUF

The Ring Oscillator PUF (ROPUF) PUF uses multiple oscillators which feed the counters. Figure 2.2 depicts the basic PUF circuit having two oscillators. For this circuit to work as a PUF it is required that the two oscillators have the identical implementation on silicon so that the delay difference is only due to the process variations. Each ring oscillator oscillates with a frequency that deviates slightly from the design value based on process variations. The output of the two oscillators is fed to two separate counters and after a certain period of counting the output of the two counters is compared to produce a 0 or a 1 PUF response bit. A RO-based PUF can have multiple ring oscillators before the counter, as shown in figure 2.3 which may be preselected or a multiplexer may be used before the counter allowing the user to

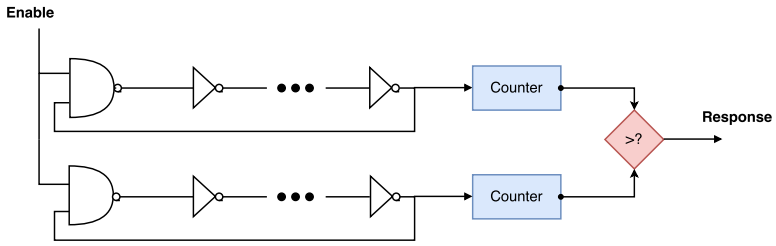


Figure 2.2: Basic Ring Oscillator PUF

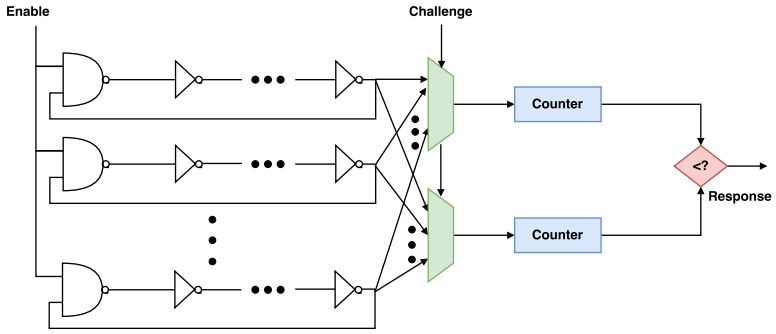


Figure 2.3: Ring Oscillator PUF using multiple ring oscillators and a counter [41]

form challenge-response pairs. In this way, this weak PUF is converted into a strong PUF by allowing the user to select the multiplexer select line bits.

2.3 Butterfly PUF

The Butterfly PUF [24] is a type of PUF targeted toward FPGAs. It consists of a pair of cross-coupled latches which tries to mimic the startup behavior of cross-coupled inverters in a SRAM cell. The Butterfly PUF (BPUF) circuit is shown in Figure 2.4. These latches contain both preset and clear signals, both of which are asynchronous. The response of the PUF is generated by sending an excitation signal that triggers the preset signal of one latch and the clear signal of the other, which makes the BPUF circuit enter an unstable state. The circuit is then allowed to settle to one of the two stable states that are possible. When the excitation signal is made low after a few clock cycles, the BPUF starts to attain a stable state. This state depends on the delay differential of the interconnects which are designed using

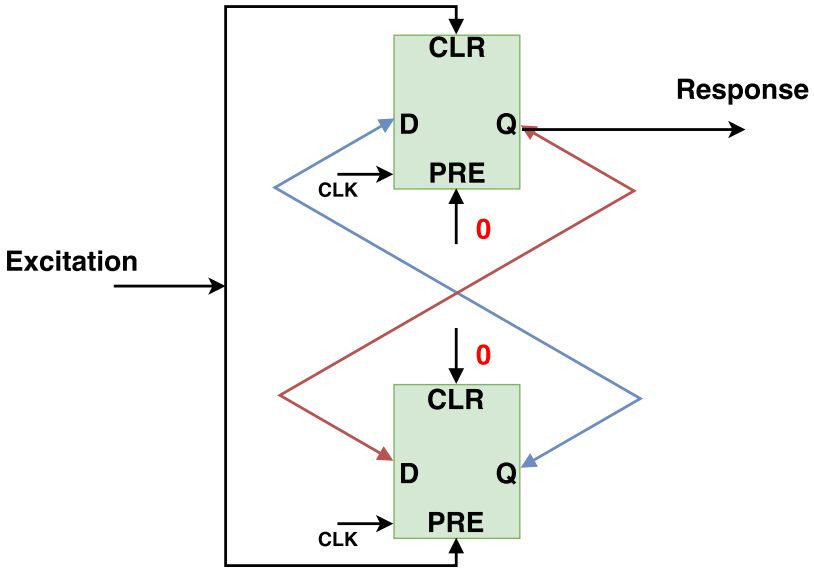


Figure 2.4: Butterfly PUF [24]

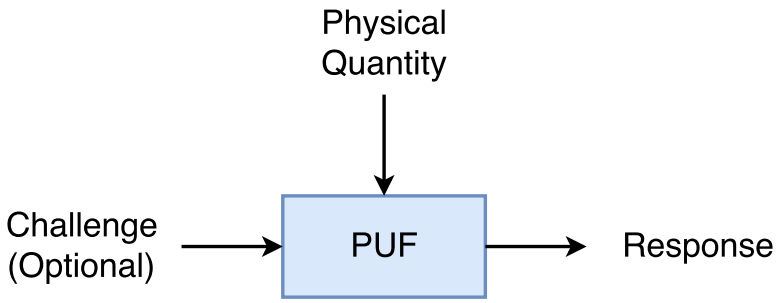


Figure 2.5: Sensor based PUF [33]

symmetrical paths on the FPGA matrix [24]. This PUF is an example of weak PUF as no challenge-response pairs are available.

2.4 Sensor based PUF

The sensor-based PUFs are a more recent type of PUFs. These PUFs use pre-existing sensors in the device to generate a random key. Figure 2.5 shows the basic structure of a sensor based PUF. The sensor can be any transducer that can sense ambient physical quantity. The PUF can be either weak or strong depending on whether it can accept a challenge or not. Depending on the physical quantity, which

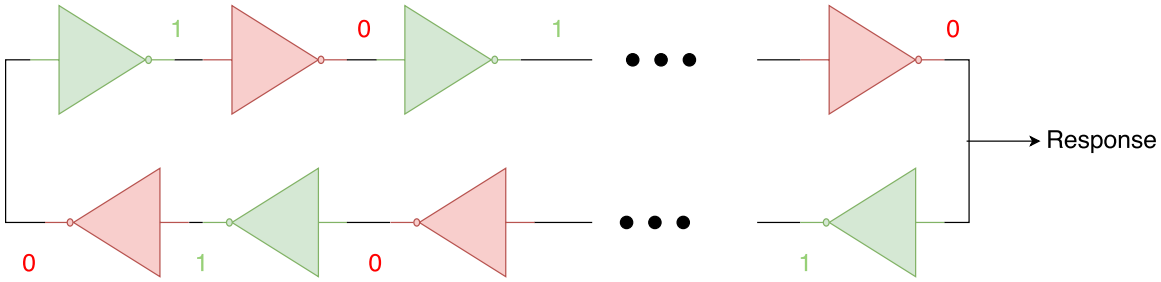


Figure 2.6: Basic bistable ring

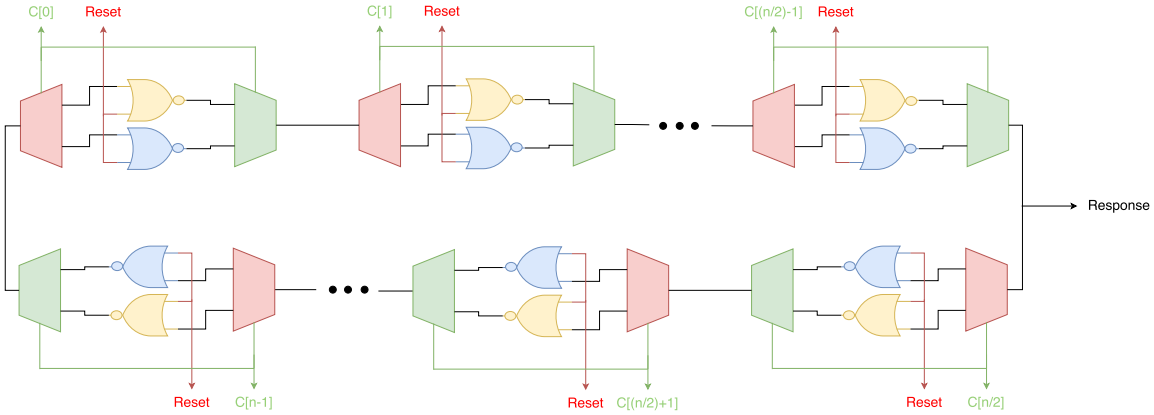


Figure 2.7: Bistable Ring PUF[5]

can be temperature, pressure, light intensity or any other quantity, the PUF response varies. In one prior work [43], a MEMs based PUF using a Gyroscope sensor is developed. The PUF is for key generation using helper data [9] stored within the ASIC chip.

2.5 Bistable Ring PUF

This is another type of PUF which is very similar in structure to ring oscillator PUF, but instead of having odd elements in the chain this PUF has an even number of inverting elements, making it bistable. Fig. 2.6 shows a logical view of Bistable ring PUF [5, 46, 6].

To make the PUF a strong PUF and to make it resettable, the structure shown in Fig. 2.7 is used. The PUF consists of n blocks. Each block contains 2 NOR gates, one

multiplexer, and one demultiplexer. The NOR gates act like inverters when reset is low, otherwise when reset is high the output of each gate is set to zero. The challenge bits C[0] to C[n-1] are applied to the select lines of multiplexers and demultiplexers as shown in the figure. The elements of the chain are thus selected according to the applied challenge C. Based on the strength variations of the NORs, a 0 or 1 is produced at the output.

2.6 Anderson PUF

The above-discussed PUF designs are not targeted specifically for FPGA implementation and are rather targeted towards custom IC implementation. These designs pose problems when implemented on FPGA. One problem that comes in for the PUFs discussed above is that they require the logic and routing to be identical along the delay paths, which guarantees that the delay arising in the output is solely due to the process variations and not due to logic or routing bias. Implementing identical delay paths is rather complicated in FPGA and requires the use of hard macros, which incorporate fixed placement and routing. The use of hard macros complicate the design and requires manual labor from the designer. The designer has to manually balance the path delays to make these instances work as PUFs. Manual routing is tedious and is subject to errors. Furthermore, hard-coded macros may result in routing that may obstruct other design signals in the routing stage of the flow, potentially increasing design congestion and reducing circuit performance. Finally, the prior PUF designs consume considerable silicon area per PUF bit. Anderson PUF[3] deals with these issues elegantly and requires the designer to only work at the behavioral level.

The basic circuit of the Anderson PUF is shown in 2.8. This is a novel PUF that doesn't require the use of hard macros. The design is done purely at the behavioral level. This PUF uses SLICEM cells in Xilinx FPGA in which there are components

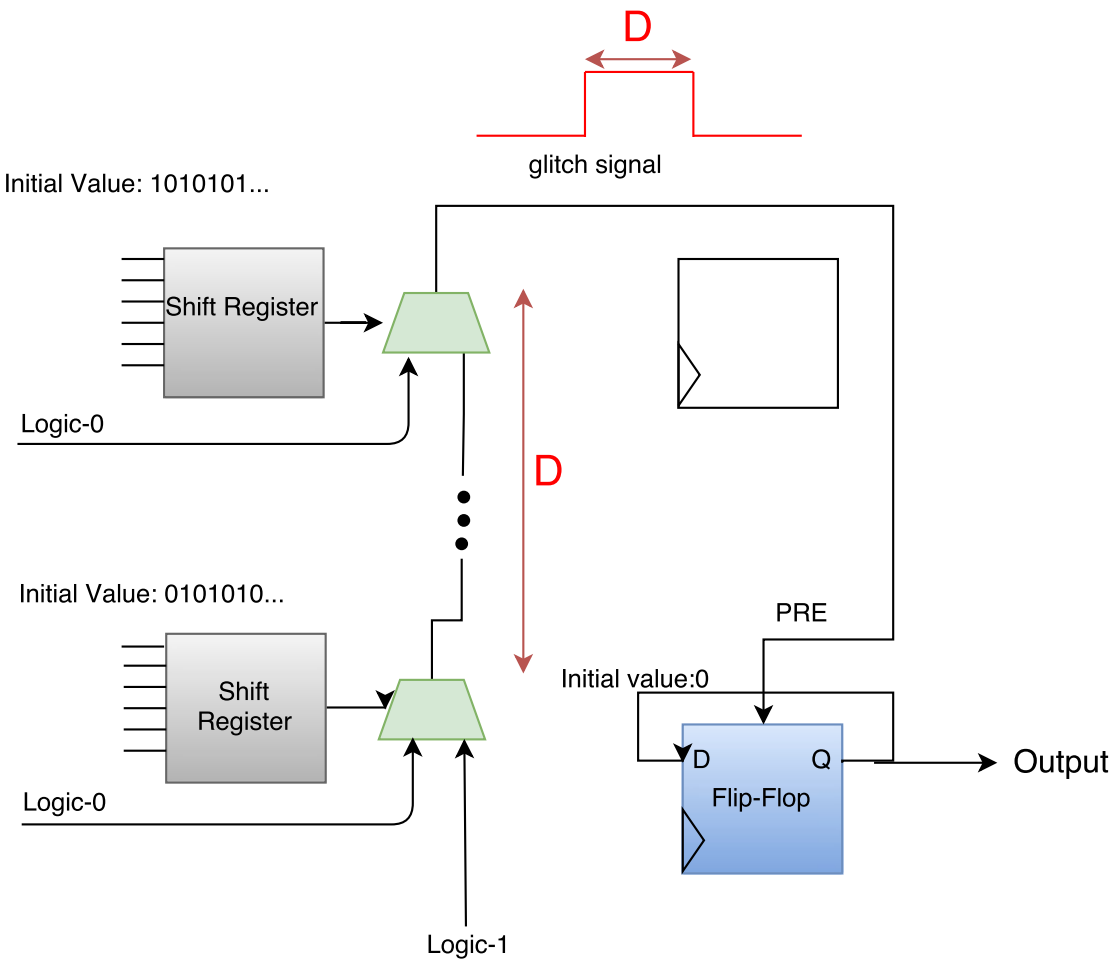


Figure 2.8: Anderson's PUF on Xilinx FPGA[3]

that can be used as LUT for combinational logic or as shift registers for memory. The length of the shift register in one instance is 16-bit.

In this PUF, the shift registers (shown in grey) are initially loaded with complemented alternating 0-1 values and then the outputs of these shift registers are fed to the select lines of the multiplexers (shown in green) in the carry chain. The transitions at the input of the select line of the multiplexer can cause a glitch to be produced at the end of the carry chain. The width of the glitch is proportional to the delay of the carry chain. If the glitch is small it is filtered out in the routing wire, otherwise, it reaches the asynchronous preset of the flip-flop (shown in blue) and sets the output

of the flip-flop to logic 1. The output of this flip-flop is taken as the output response of the PUF. This implementation of Anderson PUF requires shift registers and can only be implemented in SLICEM cells of the FPGA chip.

2.7 Challenges associated with PUFs

While both strong and weak PUFs can be used for authentication purposes, both come with their disadvantages. The strong PUFs are susceptible to modeling attacks using machine learning algorithms. This is because of a large pool of challenge-response pairs available to perform training. Attacks against strong have been performed and shown to predict the response of the PUF with very high accuracy [36, 18, 37]. Weak PUFs, on the other hand, have very limited set of challenge-response pairs or in extreme cases just one. They are not susceptible to modeling attacks because the response of the weak PUF is kept secret throughout its lifetime. Thus to use these PUFs we need to store helper data in the system to generate keys to perform error correction and key generation. This is discussed in detail in chapter 6. The risks of using PUFs have been discussed further in [22].

CHAPTER 3

STATIC RANDOM ACCESS MEMORY BASED PHYSICAL UNCLONABLE FUNCTION

This chapter explains the use SRAM cells as PUFs. The SRAM PUF [14] [12] utilises the conventional SRAM cell to generate its response. The SRAM PUF is a weak PUF when it's power-up response is used as a fingerprint. We can modify this PUF to function act as a strong PUF as discussed in [16]. We present our analysis of the PUF in terms of reliability and uniqueness using the 8Kx8 bit SRAM chip AS6C6264.

3.1 SRAM design

The SRAM cell usually consists of six transistors, four of which are used in making a cross-coupled inverter for regenerative feedback. The two transistors M1 and M3, shown in fig. 3.1, are called the access transistors and connect the inverters to the bit-lines for writing or reading to the cell. The access transistors are controlled through the word line. The word line selects the word which has to read or written according to the operation needed. Before the read or write operation, the bit-lines BL and $\overline{BL}$ are precharged. In the case of a read operation, the following steps are performed:

1. The precharge and equalization circuit is activated by raising the precharge clock high. This causes the BL and $\overline{BL}$ to rise to to a certain precharge voltage.

2. The word to be read out is selected by turning on its word line WL. The data stored in the cell causes a differential in the voltage levels of the two-bit lines according to the value stored on it.

3. Once an adequate difference is generated, the sense amplifier is turned on by turning on the transistors M1 and M2. The small difference in the input on the two inverters is amplified due to regenerative feedback and the bit lines are pulled to VDD or ground respectively.

The write operation is done similarly by performing the following steps:

1. The inputs are applied to the bit lines through a strong driver. To write a 1 in the cell, a 1 is applied to BL and 0 to $\overline{BL}$.

2. The word to be written selected by turning on its word line WL. The input of the cell inverters gets pulled up or down by the bit lines.

3. Due to the regenerative feedback of the cross-coupled inverters, the value is quickly settled to the values at the bit lines.

3.2 SRAM PUF operation

The basic six transistor SRAM cell is shown in figure 3.2. It consists of two cross coupled CMOS inverters along with two NMOS access transisitor. Each cell can store 1 bit of information. The initial state of the SRAM cell after powerup can be used as PUF to generate random and unique signatures. The response of the SRAM cell depends upon the relative strengths of the two cross coupled CMOS inverters. Initially, when no power is applied, the output of both the inverters Q and $\bar{Q}$ is a 0. When power is applied, the transistors turn on and try to pull up their outputs to a high value. This creates metastability in the cell as the inverters are cross coupled. The regenerative feedback of the inverters accelerates the settling of the response of the cell. The stronger pull-up in the cell settles to a output of 1 while a stronger pull down settles to a 0. The affinity of the cell to settle to 1 or a zero response is termed as its skew.

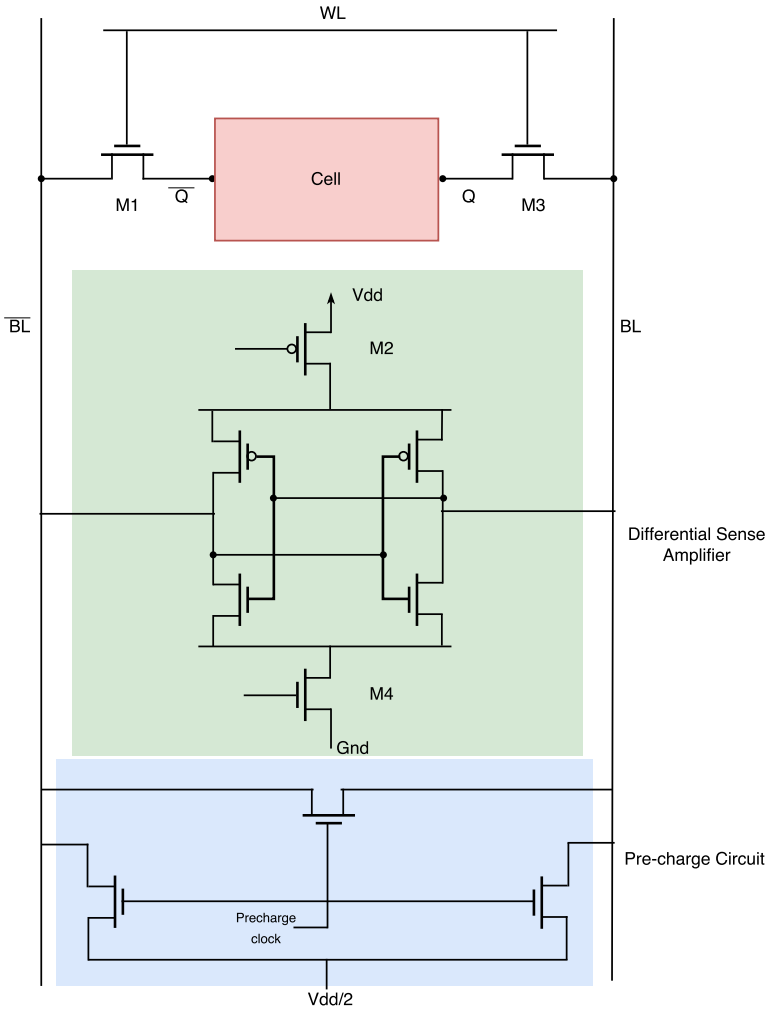


Figure 3.1: SRAM chip circuitry [38]

The skew of a cell is subject to both the process variations and the noise at start-up. The power-up state of the cell is directly dependent on the skew of the cell. Skew at a given power-up is influenced by noise, so the skew of each cell across many power-ups is described by a probability distribution function (Fig. 3.3). A skewed cell (Fig. 3.3b) always powers up to a 0 or 1 depending upon the direction of the skew and is the effect of noise is not enough to flip the response. On the other hand, a non-skewed or neutral-skewed cell (Fig. 3.3a) can power up to either 0 or 1, depending on the noise conditions. Although a non-skewed cell would seem to be composed of matched devices, this may not be always the case, but rather different

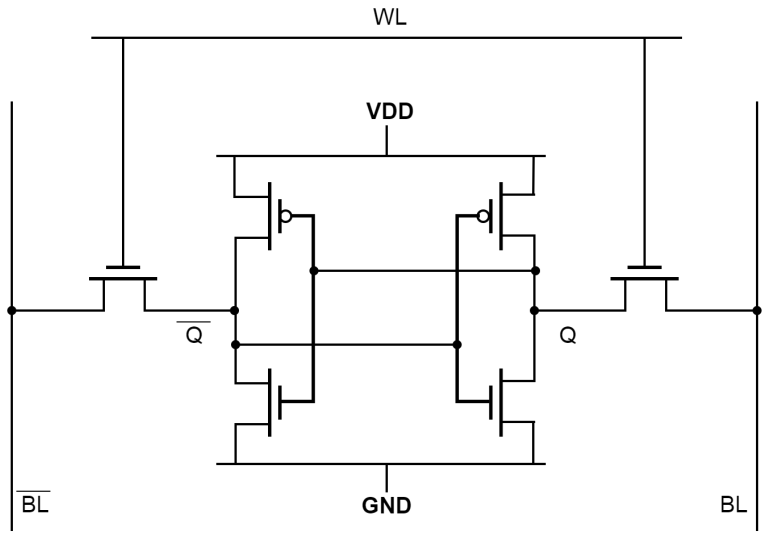


Figure 3.2: SRAM cell [15]

(a) Unskewed SRAM cell

(b) 0-skewed SRAM cell

Figure 3.3: Figure showing effect of process variations and noise on SRAM cell

process variations canceling out each other to create a non-skewed behavior. This behavior, therefore, might vary as the operating conditions are varied [15, 17].

3.3 SRAM chip architecture

Three different chips are used in our experiments. The specifications of these chips are tabulated in table 3.1. The architecture of one of the chip (AS6C6264) is shown in figure 3.4. The chip has a capacity of 8Kbytes, each word being 8-bit wide. The Chip has a parallel interface for addressing the words. The decoder selects one of the word lines based on the address provided to be read or written in the chip array. The

Table 3.1: SRAM chips under evaluation

IC	Manufacturer	Capacity	Fabrication technology	Interface
AS6C6264	Alliance memory	8 KB	350 nm	Parallel
23LC1024	Microchip	128 KB	160 nm	Serial SPI
CY7C185	Cypress Semiconductor	8 KB	90 nm	Parallel

control circuitry block controls the operation to be performed on the chip. The I/O data circuit is a bidirectional bus which can output the data from the cell as well as take the data from the outside driver for writing inside the chip.

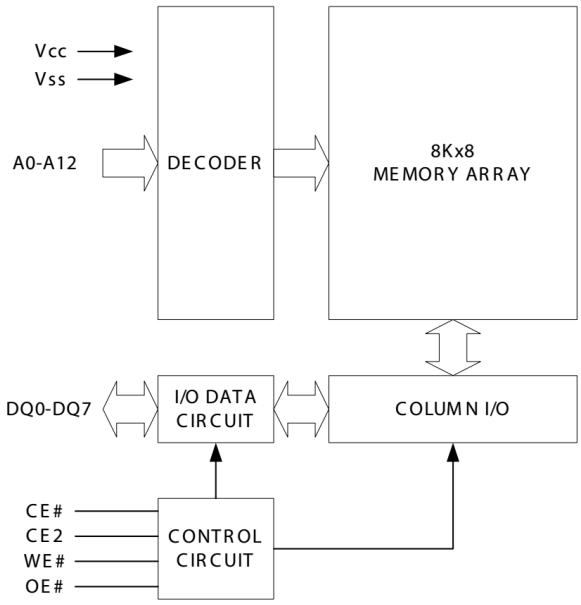


Figure 3.4: AS6C6264 SRAM chip details[2]

The other chips also have similar architectures. Readers are referred to [31] and [8] for details.

3.4 Experimental validation

A primary research goal is to determine if the SRAM PUF can be used to generate temperature specific keys. The PUF performance has to be measured at different temperatures to quantify the change in response. The PUF performance is defined

by two primary factors, uniqueness (Within class Hamming distance) and reliability (Between class Hamming distance).

For our analysis, we evaluated the power-up response of the three SRAM chips. Chips AS6C6264 and CY7C185 have 8K locations each having a word size of 8-bit. Hence, a total of 65536-bits can be generated. For chip 23LC1024, we have 1Mbit of PUFs. We evaluated the three different chips. In this section, we describe the experiments used to analyze these parameters. We quantify the two properties of the PUFs by dividing all the PUF instances into blocks of 128 PUFs bits across the chip for all the three chips and their instances. In total, we have 512 disjoint 128-bit PUFs on chips AS6C6264 and CY7C185 and 8192 disjoint 128-bit PUFs on chip 23LC1024.

3.4.1 Uniqueness

PUFs are used generate device-specific fingerprints. To identify each device uniquely, each instance of the PUF must produce a response that is independent and different from the response of the other PUF instances. To measure the randomness of responses across instance we calculate the uniqueness parameter for the PUF. The uniqueness is calculated by computing the Hamming distance between the response of the two PUF instances [10]. The Hamming distance for any two n-bit output responses O_a and O_b is calculated using equation 3.1.

$$HD(O_a, O_b) = \sum_{i=0}^{n-1} (O_a[i] \bigoplus O_b[i]) \tag{3.1}$$

To calculate the uniqueness or Between class Hamming distance we use eq. 3.2. The ideal uniqueness for any PUF is 50%, meaning that half the bits of the response are always different.

$$BHD(i, j) = \frac{1}{\frac{k*(k-1)}{2}} \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} HD(O_{i,a}, O_{j,b}) \tag{3.2}$$

Here i and j are the n-bit PUFs under comparison and k is the number of trials. $O_{i,a}$ represents the output response of PUF instance i in a^{th} trial. Since we consider 128-bit outputs, the ideal between class Hamming distance in our case is 64 bits.

3.4.2 Reliability

For PUFs to act as device authentication entities or to generate keys, their responses must be repeatable over evaluations. To measure the repeatability of the responses of a PUF instance, we calculate its Within class Hamming distance. Within class hamming distance measures the Hamming distance between two trials of the same PUF. The ideal value of within class Hamming distance is 0, meaning that the responses are perfectly repeatable. Due to noise and variation in operating conditions, this ideal value is not achievable. The equation for calculating the average Within class Hamming distance given by eq. 3.3.

$$WHD = \frac{1}{\frac{(k-1)*(k-2)}{2}} \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} HD(O_i, O_j) \tag{3.3}$$

Here n is the length of the response, k is the trial number and O_i represents the response of the PUF in the i^{th} trial. This equation calculates the Hamming distance of all possible combinations of the trials of the PUF.

3.5 Results and analysis

The between class Hamming distance obtained by comparing two 128 bit PUFs in the same locations from random chips and randomly selected output trials. Over 1000,000 comparisons the between class Hamming distance is shown in red in Figures 3.5, 3.6 and 3.7 for the 3 chips under evaluation. The within class Hamming distance measuring the reliability of the 128-bit PUF instances is shown in blue in figures 3.5, 3.6 and 3.7. The results of these experiments are tabulated in table 3.2. From these

Table 3.2: PUF reliability and uniqueness for 3 different SRAM chips evaluated using 128-bit PUF blocks

IC	Within class Hamming distance	Between class Hamming distance
AS6C6264	4.24 bits (3.38%)	63.02 bits (49.22%)
23LC1024	9.33 bits (7.28%)	53.05 bits (41.44%)
CY7C185	11.44 bit (8.90%)	57.72 bits (45.10%)

results, we can conclude that the PUFs produces a unique outputs. Also, the PUFs have low within class Hamming distance showing that the PUFs are reliable.

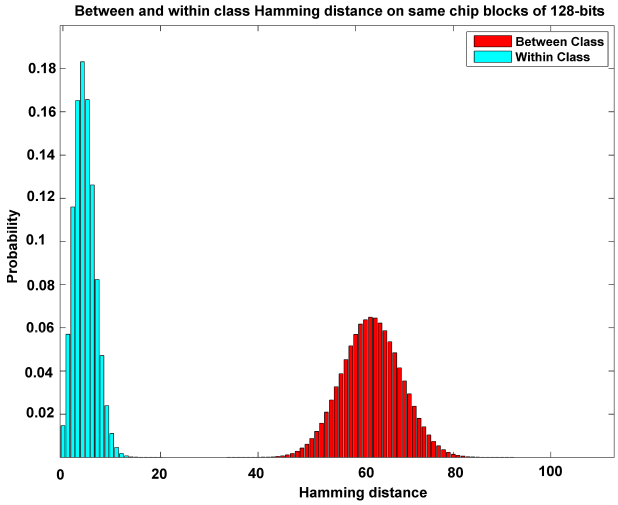


Figure 3.5: Between and within class Hamming distances for 128 bit PUFs from AS6C6264 chips

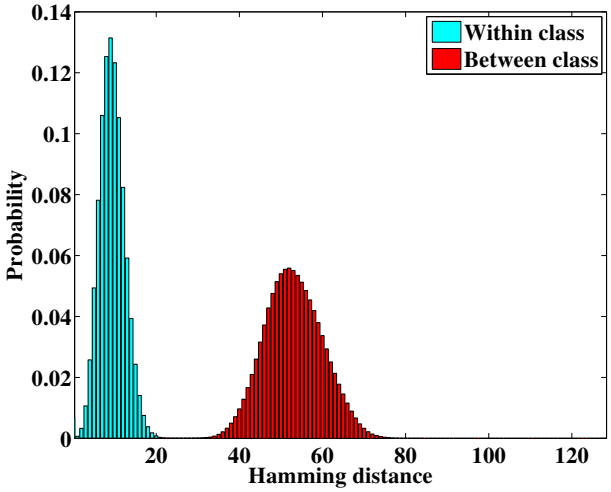


Figure 3.6: Between and within class Hamming distances for 128 bit PUFs from 23LC1024 chips

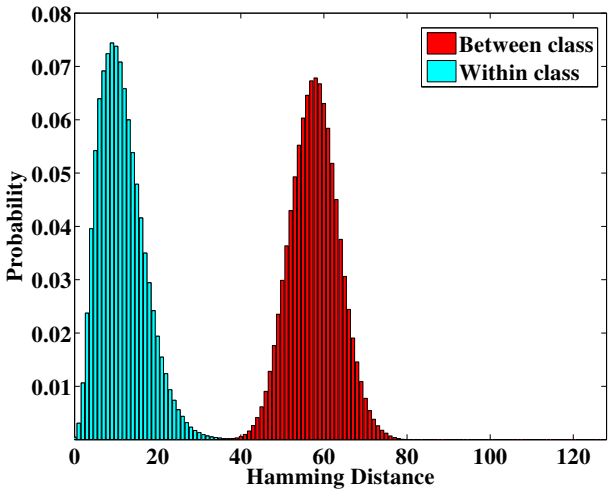


Figure 3.7: Between and within class Hamming distances for 128 bit PUFs from CY7C185 chips

CHAPTER 4

PUF BASED KEYS AS PROOFS OF TEMPERATURE

In this Chapter, we first give a detailed description of the methodology for data collection and the data collection system. We then propose the idea of using SRAM PUFs along with the temperature of the environment as a means to establish key between two communicating nodes. We provide evidence to support our approach by calculating the uniqueness of the PUFs on all the locations of a chip and observing the shift in between temperature Hamming distance of the PUFs with respect to a minimum temperature of 0°C. Also, we discuss the error correcting codes that will be used in our work along with a full system design to generate and establish temperature based key by utilising temperature sensor along with temperature dependent bits generated by the PUF corrected by the error correcting codes.

4.1 Main objective

The objective of this research is to use PUFs to generate keys which change with temperature with fixed step size. The response of the PUFs changes with temperatures. If the distance between the responses at two different consecutive temperatures is higher than the within class (temperature) hamming distances at those temperatures, then we can exploit this variation to generate temperature dependent keys. We will have to explore the minimum temperature difference between responses that generates a Hamming distance greater than the within temperature Hamming distance. This is important because if the Hamming distance of the PUF responses between two temperatures is not big enough, a noisy trial from one temperature can be considered

as the PUF output from a different temperature. Once we find a temperature step size that produces sufficiently large between temperature Hamming distances, we can use it to generate keys unique to those temperature steps.

Since the key for a particular temperature step is unique and can only be generated when the ambient temperature of the system in the range of that temperature step, these unique keys can be used as an indicator of temperature. Thus a temperature sensor is created that cannot be forged.

4.2 Data Collection

4.2.1 Data remanence

To collect multiple powerup responses from the SRAMs, we need to evaluate the maximum retention time at lowest temperature (0°C) under evaluation. This is needed to make sure that the old values from the SRAMs are erased before evaluating a new trial. Memories are devices that store the state in the form of charges. These charges are lost due to leakages in the devices causing the data to be lost when power is turned off. At low temperatures, the leakages are reduced exponentially. The leakage current in an MOS device is directly proportional to the temperature as shown in the equation 4.1. Due to the reduced leakage current, the time required for the discharge of state increases exponentially. Due to this phenomenon, the memories are subject to cold boot attacks[13]. Some other attacks and prevention methods are discussed in [48] and [32].

The subthreshold leakages are given by equation

$$I_{subthreshold} = A_s \frac{W}{L} v_T^2 (1 - e^{\frac{-V_{DS}}{v_T}}) e^{\frac{V_{GS}-V_{th}}{n v_T}} \tag{4.1}$$

- where A_s is a technology-dependent constant,
- V_{th} is the threshold voltage,

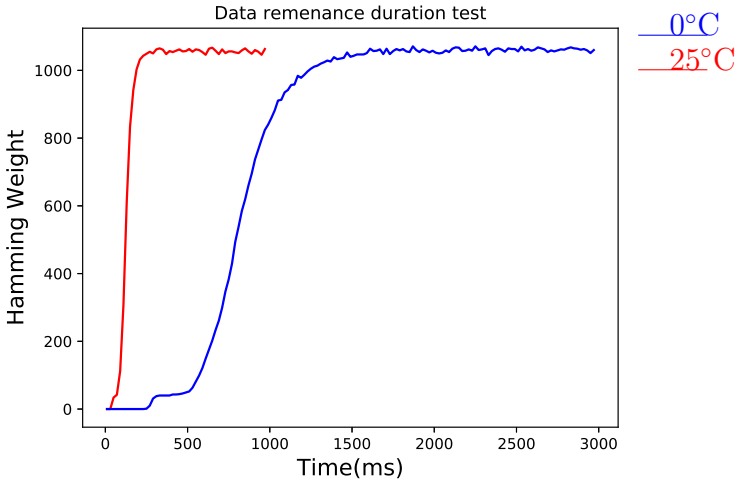


Figure 4.1: Data remanence duration after power down at different temperatures

- L and W are the device effective channel length and width,
- V_{GS} is the gate-to-source voltage,
- n is the subthreshold swing coefficient for the transistor,
- V_{DS} is the drain-to-source voltage, and
- v_T is the thermal voltage.

According to equation 4.1, the leakage and temperature have quadratic relationship.

To test the data retention of our SRAM chip, the following steps were performed:

1. The SRAM chip is soaked at the desired temperature in the heat chamber for 7 minutes.
2. A microcontroller writes in zeroes in all the cells of the SRAM chip.
3. The power line of the chip is grounded and then the data is read out after a specified delay.
4. Delay is incremented by 20ms and experiment is repeated from step 2 until the Hamming weight stops increasing further.

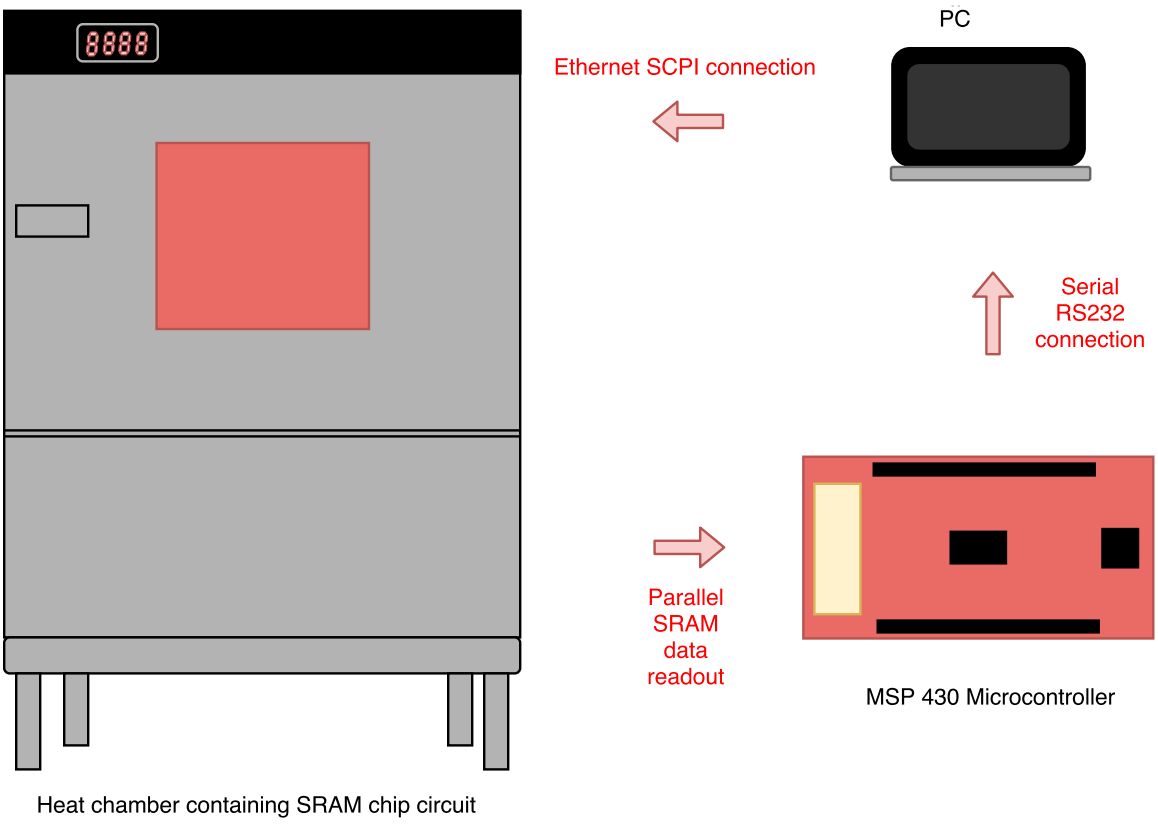


Figure 4.2: System for data collection at different temperatures

Following the approach discussed above, we generated the retention times for two temperatures as shown in figure 4.2.1. It can be observed that at 0 degrees the retention time is much higher that at 25 degrees. To collect the response at different temperatures from the SRAM chip, we have to abide by the minimum power off duration found from the curve, so that the previous power-up state is completely erased from the chip.

4.2.2 Data-collection setup

Figure 4.2 shows the basic setup used to extract data off the SRAM at different temperatures. We collect the data over a range of 0°C to 55°C. The following steps are performed to generate 100 trials from a single chip at a single temperature:

1. The PC runs a python script which communicates with the heat chamber over ethernet to control the temperature. The script sets the initial temperature to 0^oC.

2. The system waits for 7 minutes of soak time to make sure that the chip is at the same temperature as the ambience.

3. After completion of soak time, a command is sent to the TI MSP430 microcontroller to begin reading the data from the SRAM chip.

4. The Microcontroller collects 100 trials by powering the chip on and off using BJTs. The power off time is dictated by remanence time as discussed in section 4.2.1. It then sends the data read over through serial RS-232 link to the PC which gets logged in a file.

5. The temperature is increased by a step of 5°C process is repeated until 55°C is reached.

4.2.3 Temperature response of SRAM

In this section, we present our temperature results generated from the AS6C6264 SRAM chip. Figures 4.3 show the variation in the response of the SRAM PUF at different temperatures. The Hamming distances are calculated by setting the 0^oC response as the base response. This figure clearly shows a positive drift in the Hamming distances as the temperature is increased. To allow for easy distinction of temperatures, majority voting is performed on the data and Hamming distances are recalculated. We can clearly see distinct histograms for different temperatures.Tables 4.1 and 4.2 tabulates the between Hamming distances for chip 1 between all temperature pairs. We see the same pattern of increasing distance from temperature increases. Also, in table 4.2 we see an increase in within temperature distance, which makes it easier to distinguish between two temperatures. We only include one more

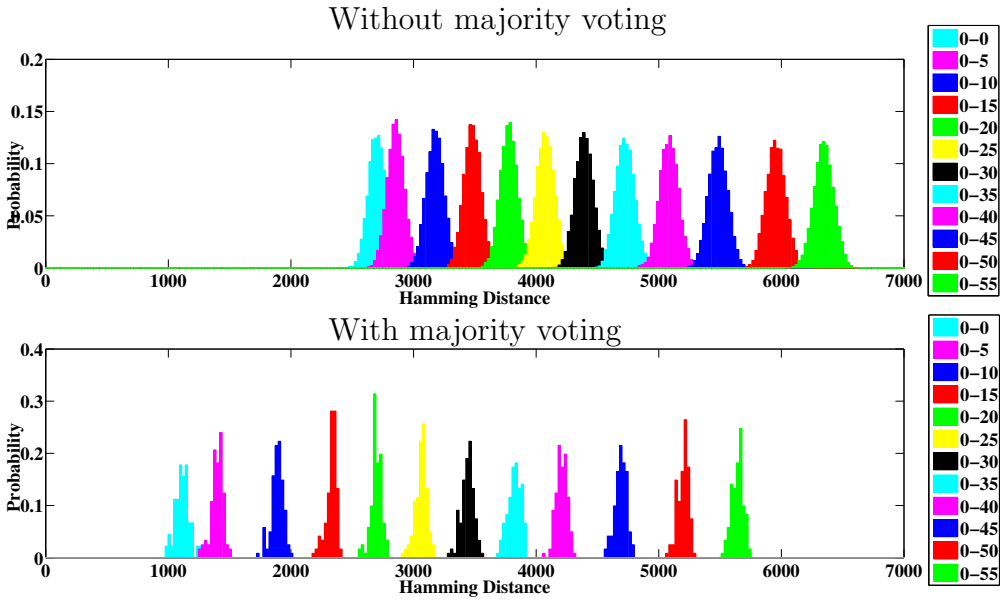


Figure 4.3: BER of PUF instances placed on AS6C6264 chip 1 across different temperatures

chips data (Figure 4.4) to show that this phenomenon is general and is applicable for all chips, though it occurs for all the 10 chips under evaluation.

4.2.4 Testing on advanced technology SRAM

In this section we present the results taken from two advanced technology SRAM chips. The first one is a Microchip 23LC1024 [31] which is a 128KB SRAM chip with SPI interface fabricated in 160nm technology.The second is a Cypress CY7C185 [8] which is an 8KB SRAM chip with parallel I/O and is fabricated in 90nm technology. We performed the temperature data collection and analysis experiments to test whether this technique is feasible with newer technology SRAM cells. All the results are generated from 8KB responses (65536-bits responses). Figures 4.5 and 4.6 show the temperature responses of 2 chips respectively. It is clear that the response follows a similar trend as the results from the 350nm SRAM chip described in sec 4.2.3. Tables 4.3 and 4.5 show the change in Hamming distance with temperatures

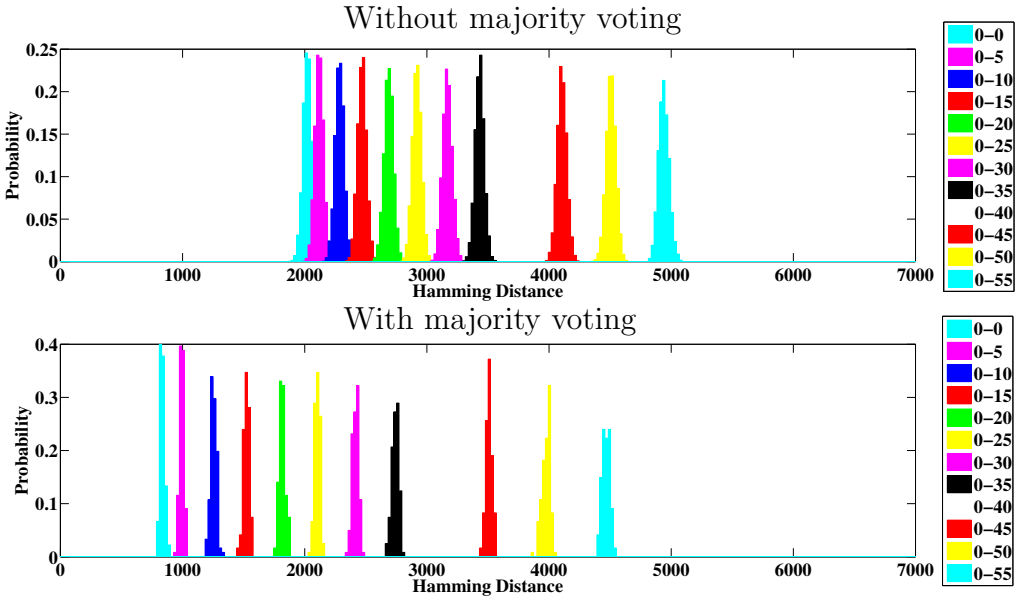


Figure 4.4: BER of PUF instances placed on AS6C6264 chip 2 across different temperatures

Table 4.1: Between temperature average Hamming distances observed on 1 chip

Temp(C)	0	5	10	15	20	25	30	35	40	45	50	55
0	4.13	-	-	-	-	-	-	-	-	-	-	-
5	4.37	4.20	-	-	-	-	-	-	-	-	-	-
10	4.86	4.44	4.27	-	-	-	-	-	-	-	-	-
15	5.33	4.80	4.45	4.34	-	-	-	-	-	-	-	-
20	5.78	5.21	4.79	4.53	4.42	-	-	-	-	-	-	-
25	6.23	5.66	5.20	4.87	4.62	4.51	-	-	-	-	-	-
30	6.72	6.16	5.69	5.31	4.97	4.70	4.59	-	-	-	-	-
35	7.22	6.70	6.24	5.83	5.45	5.09	4.79	4.66	-	-	-	-
40	7.77	7.29	6.86	6.43	6.02	5.60	5.20	4.87	4.75	-	-	-
45	8.38	7.96	7.56	7.13	6.70	6.23	5.75	5.30	4.99	4.83	-	-
50	9.10	8.74	8.38	7.97	7.52	7.02	6.48	5.93	5.48	5.08	4.89	-
55	9.70	9.39	9.07	8.66	8.22	7.70	7.12	6.52	5.99	5.46	5.06	4.90

Table 4.2: Between temperature average Hamming distances observed on 1 chip after Majority voting

Temp(C)	0	5	10	15	20	25	30	35	40	45	50	55
0	1.72	-	-	-	-	-	-	-	-	-	-	-
5	2.15	1.76	-	-	-	-	-	-	-	-	-	-
10	2.90	2.14	1.74	-	-	-	-	-	-	-	-	-
15	3.57	2.73	2.07	1.85	-	-	-	-	-	-	-	-
20	4.13	3.26	2.55	2.12	1.82	-	-	-	-	-	-	-
25	4.69	3.85	3.13	2.61	2.12	1.82	-	-	-	-	-	-
30	5.26	4.49	3.81	3.25	2.68	2.15	1.88	-	-	-	-	-
35	5.85	5.16	4.53	3.97	3.37	2.75	2.21	1.89	-	-	-	-
40	6.44	5.84	5.28	4.74	4.15	3.48	2.84	2.24	1.98	-	-	-
45	7.17	6.67	6.18	5.67	5.10	4.42	3.71	2.99	2.44	2.10	-	-
50	7.95	7.56	7.15	6.68	6.14	5.46	4.73	3.92	3.22	2.50	2.05	-
55	8.63	8.30	7.94	7.51	6.99	6.33	5.59	4.77	4.00	3.13	2.35	2.01

for both the chips respectively. Tables 4.4 and 4.6 shows the values after 9-majority voting for the 2 chips.

4.2.5 Sensitivity analysis

The sensitivity is a measure of the percentage change in the Within class Hamming distance per degree Celsius. The sensitivity is calculated using the equation 4.2. Here, T_n and T_{n+1} represents the responses of the PUF at temperature step n and n+1 respectively. In our case, the step size in 5°C. The sensitivity of the Within class Hamming distance with respect to temperature for 3 different technology SRAM cells is shown in figure 4.7. We can see that for all the SRAMs the sensitivity is pretty consistent across temperatures and stay in the range of 0.6% and 2.0%.

$$Sensitivity(T_{n+1}, T_n) = \frac{BHD(T_{n+1}, T_n) - WHD(T_n)}{WHD(T_n) * step} \tag{4.2}$$

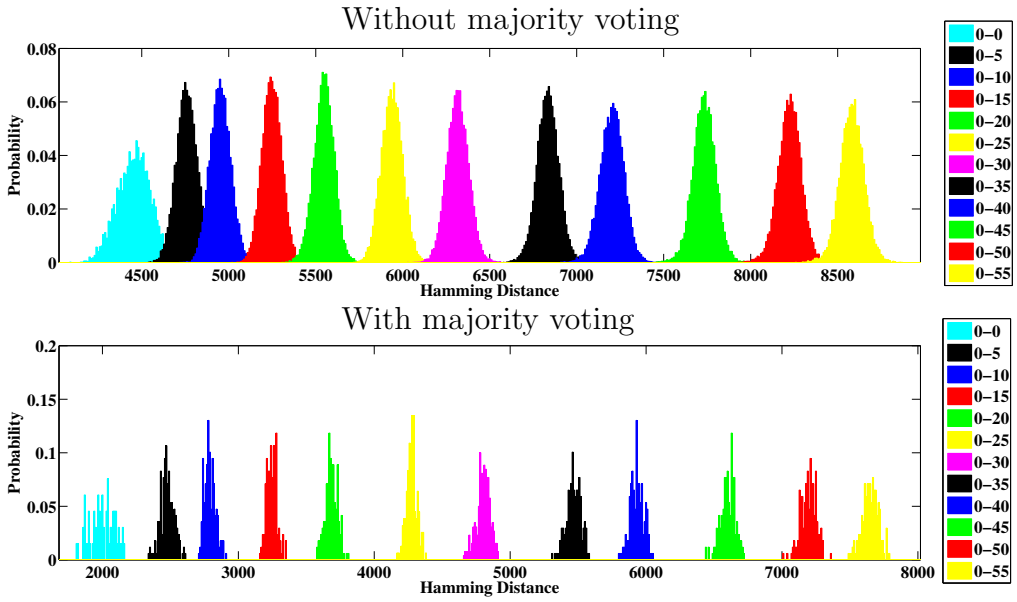


Figure 4.5: BER of PUF instances placed on chip 23LC1024 across different temperatures

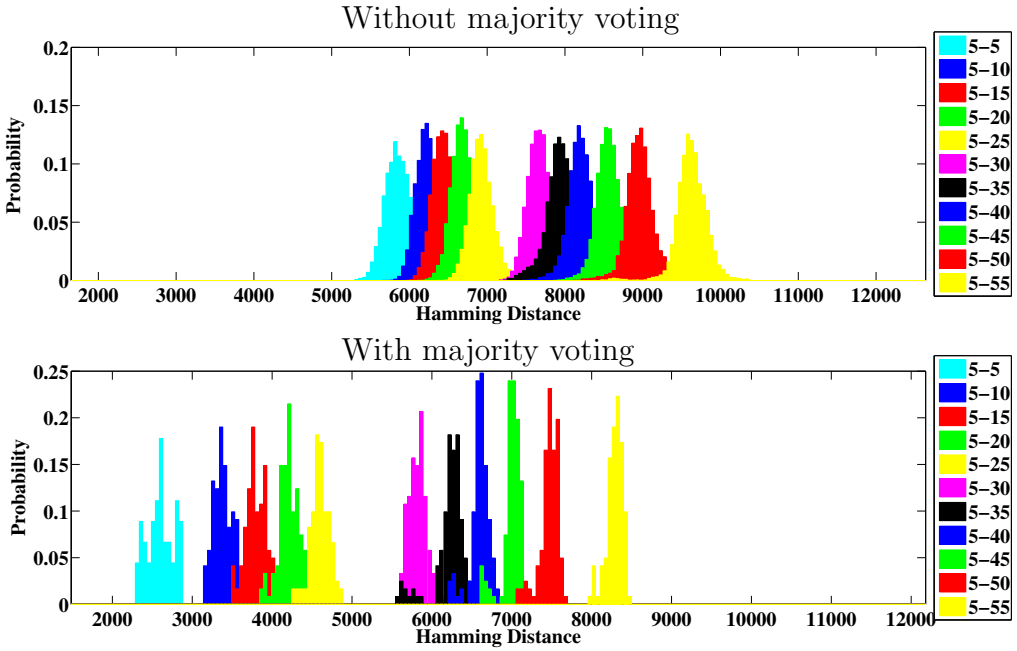


Figure 4.6: BER of PUF instances placed on chip CY7C185 across different temperatures

Table 4.3: Between temperature average Hamming distances observed on an 160nm 23LC1024 chip

Temp(C)	0	5	10	15	20	25	30	35	40	45	50	55
0	6.807	-	-	-	-	-	-	-	-	-	-	-
5	7.268	6.901	-	-	-	-	-	-	-	-	-	-
10	7.564	7.270	7.080	-	-	-	-	-	-	-	-	-
15	8.015	7.736	7.582	7.196	-	-	-	-	-	-	-	-
20	8.472	8.110	7.855	7.517	7.330	-	-	-	-	-	-	-
25	9.069	8.652	8.322	7.964	7.773	7.282	-	-	-	-	-	-
30	9.649	9.198	8.813	8.420	8.159	7.851	7.602	-	-	-	-	-
35	10.436	9.961	9.524	9.086	8.752	8.402	8.140	7.554	-	-	-	-
40	10.998	10.495	10.028	9.542	9.154	8.728	8.400	7.909	7.609	-	-	-
45	11.807	11.295	10.795	10.284	9.860	9.373	8.998	8.509	8.255	7.588	-	-
50	12.556	12.026	11.527	11.002	10.542	10.027	9.610	9.094	8.787	8.335	7.537	-
55	13.102	12.558	12.033	11.496	11.005	10.454	9.987	9.412	9.043	8.508	7.826	7.563

Table 4.4: Between temperature average Hamming distances observed on 160nm 23LC1024 chip after Majority voting

Temp(C)	0	5	10	15	20	25	30	35	40	45	50	55
0	3.08	-	-	-	-	-	-	-	-	-	-	-
5	3.78	3.16	-	-	-	-	-	-	-	-	-	-
10	4.26	3.66	3.29	-	-	-	-	-	-	-	-	-
15	4.96	4.40	4.00	3.30	-	-	-	-	-	-	-	-
20	5.63	4.98	4.45	3.76	3.40	-	-	-	-	-	-	-
25	6.53	5.85	5.23	4.53	4.10	3.17	-	-	-	-	-	-
30	7.33	6.63	5.98	5.27	4.74	4.12	3.62	-	-	-	-	-
35	8.34	7.69	7.02	6.30	5.71	5.07	4.50	3.50	-	-	-	-
40	9.06	8.39	7.72	6.98	6.35	5.61	4.97	4.07	3.54	-	-	-
45	10.08	9.42	8.76	8.03	7.39	6.63	5.97	5.13	4.64	3.55	-	-
50	10.98	10.33	9.69	8.99	8.34	7.56	6.91	6.09	5.56	4.84	3.48	-
55	11.68	11.02	10.36	9.65	8.99	8.20	7.51	6.61	6.00	5.15	3.98	3.51

Table 4.5: Between temperature average Hamming distances observed on a 90nm CY7C185 chip

Temp	5	10	15	20	25	30	35	40	45	50	55
5	8.99	-	-	-	-	-	-	-	-	-	-
10	9.56	8.85	-	-	-	-	-	-	-	-	-
15	9.89	9.20	8.79	-	-	-	-	-	-	-	-
20	10.24	9.60	9.13	8.68	-	-	-	-	-	-	-
25	10.72	10.12	9.70	9.20	8.94	-	-	-	-	-	-
30	11.74	11.13	10.66	10.16	9.61	7.97	-	-	-	-	-
35	12.15	11.56	11.09	10.60	10.05	8.35	7.82	-	-	-	-
40	12.56	12.01	11.54	11.05	10.51	8.88	8.31	7.96	-	-	-
45	13.11	12.62	12.17	11.69	11.17	9.62	9.11	8.56	8.02	-	-
50	13.71	13.24	12.79	12.33	11.82	10.30	9.80	9.26	8.58	8.11	-
55	14.73	14.30	13.89	13.45	12.99	11.58	11.10	10.57	9.95	9.35	8.44

Table 4.6: Between temperature average Hamming distances observed on a 90nm CY7C185 chip after Majority voting

Temp(C)	5	10	15	20	25	30	35	40	45	50	55
5	4.06	-	-	-	-	-	-	-	-	-	-
10	5.20	3.96	-	-	-	-	-	-	-	-	-
15	5.84	4.68	4.04	-	-	-	-	-	-	-	-
20	6.44	5.45	4.69	3.93	-	-	-	-	-	-	-
25	7.21	6.28	5.65	4.85	4.37	-	-	-	-	-	-
30	8.92	8.04	7.40	6.68	5.81	3.78	-	-	-	-	-
35	9.52	8.74	8.08	7.42	6.59	4.49	3.68	-	-	-	-
40	10.10	9.36	8.74	8.05	7.29	5.34	4.48	3.91	-	-	-
45	10.71	10.09	9.49	8.84	8.13	6.30	5.65	4.78	3.83	-	-
50	11.43	10.83	10.25	9.65	8.96	7.24	6.63	5.86	4.76	3.93	-
55	12.68	12.16	11.65	11.10	10.49	8.96	8.37	7.63	6.76	5.81	4.21

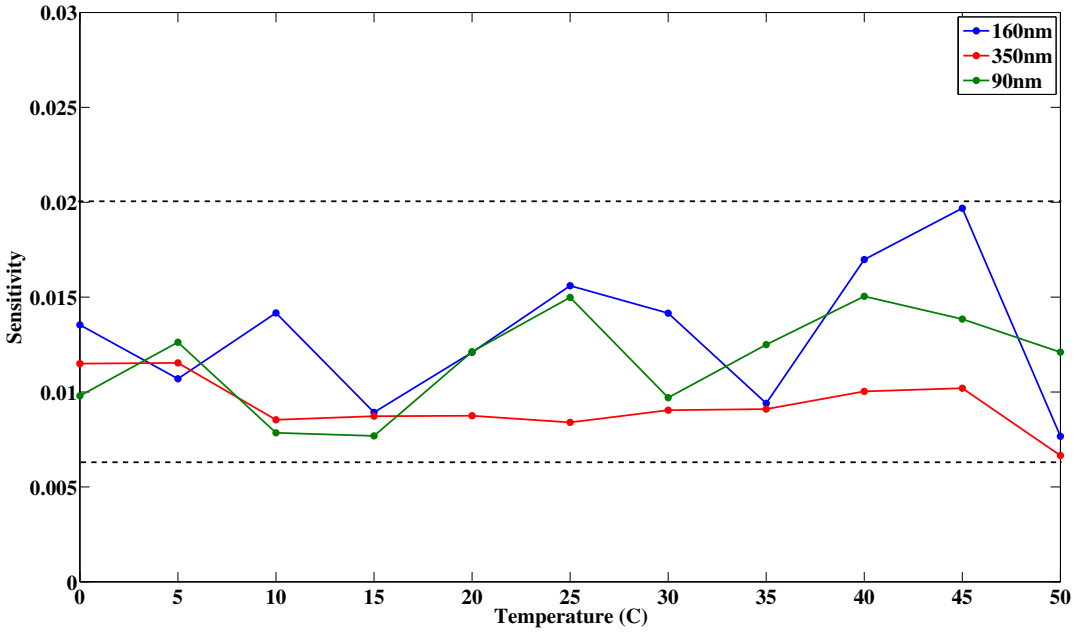


Figure 4.7: Figure showing the sensitivity of Within class Hamming distance with respect to temperatures for different technology SRAMs.

4.3 Per temperature key enrollment and generation system

This section explains the process of key establishment using a temperature sensor and temperature sensitive PUF. We then explain the importance of error correction codes and their use with temperature-based PUF keys. The process of key enrollment and generation with the help of error correction codes is described along with a detailed description of the system.

4.3.1 Key enrollment

Enrollment is done to generate Helper data from the key and the PUFs that can be used later by the system to generate the same key. Figure 4.8 shows the system for performing key enrollment. Enrollment of keys will be done for each temperature step in our case to generate a per temperature (T) key denoted by K_T. We use a BCH encoder having a block size of n-bit, message input of k-bit and t number of correctable errors. To generate a K-bit key, we have to divide it into blocks of k bits.

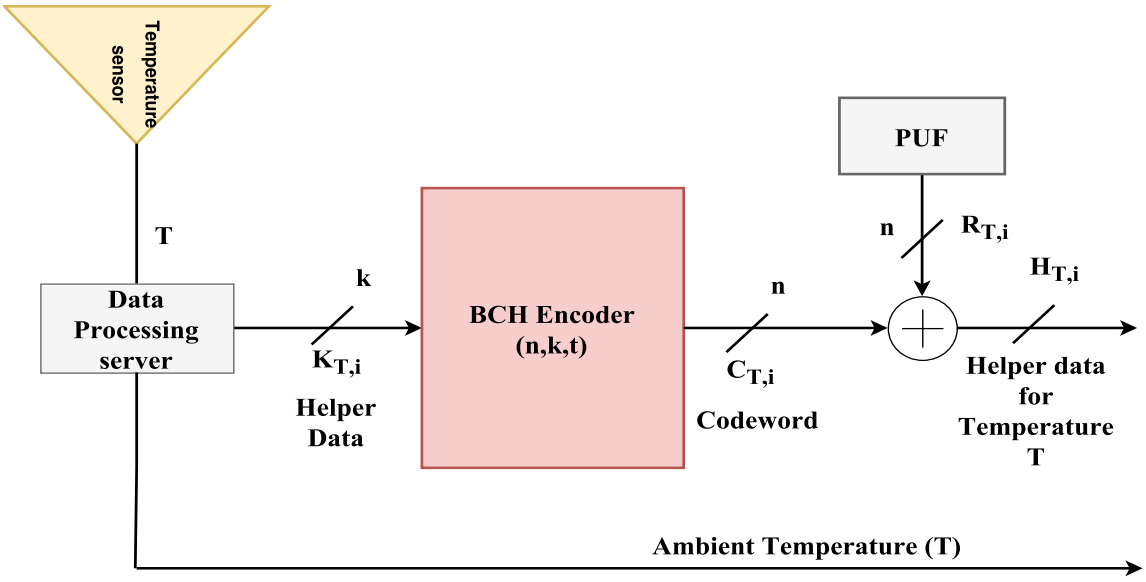


Figure 4.8: Per-temperature one time key enrollment

For each block i of k-bit input, a n-bit output codeword $C_{T,i}$ is generated by the BCH encoder. This codeword is then offset by the response of the PUF $R_{T,i}$ at the same temperature T which generates n-bit helper data $H_{T,i}$. This helper data will be used later by the system to generate the key for that temperature. The enrollment process will be performed for the whole operating temperature range of the system.

4.3.2 Key generation

Figure 4.9 shows the system used to generate the key on a per-temperature basis. To generate the key, the temperature is read from the temperature sensor and is used to read out the correct helper data $H_{T,i}$ from the memory. This helper data gets offset by the n-bit PUF response $R'_{T,i}$. The response of the PUF is noisy and tends to change over evaluations. The response used to offset the helper data may be different than the response $R_{T,i}$ used during enrollment. This will produce a corrupted codeword $C'_{T,i}$. The k-bits of the key can be recovered from the invalid codeword as long as the number of errors in the response is under the correctable errors t of the BCH

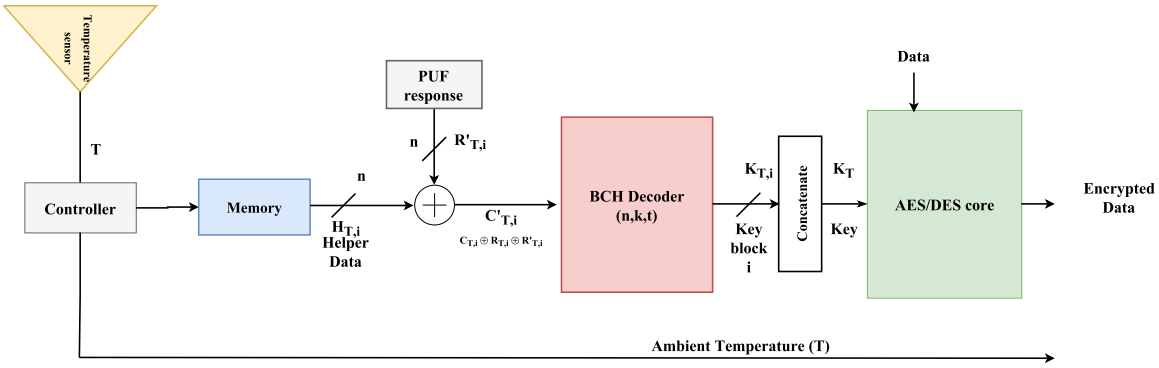


Figure 4.9: Per-temperature key generation

code used. At the output of the BCH decoder, we will get the enrolled key for that temperature if the previous condition is satisfied.

4.3.3 Complete system

The complete system example is shown in figure 4.10. When the master node requests communication with slave node, the slave node generates the key based on its ambient temperature. The key is generated using the scheme proposed in the previous section. The slave node sends its first message, containing information already known to the master, encrypted with the key generated at temperature T along with its ambient temperature to establish a secure communication line. The master node knows the keys for the slave node at each enrolled temperatures. Using the temperature reported by the slave node, the master node can use the key for that temperature to decrypt the message packet. In the case of failure in decryption of the message, the master node will be able to detect that the reported temperature was fake and the node has been tampered with. Thus, the key will serve as a proof of temperature.

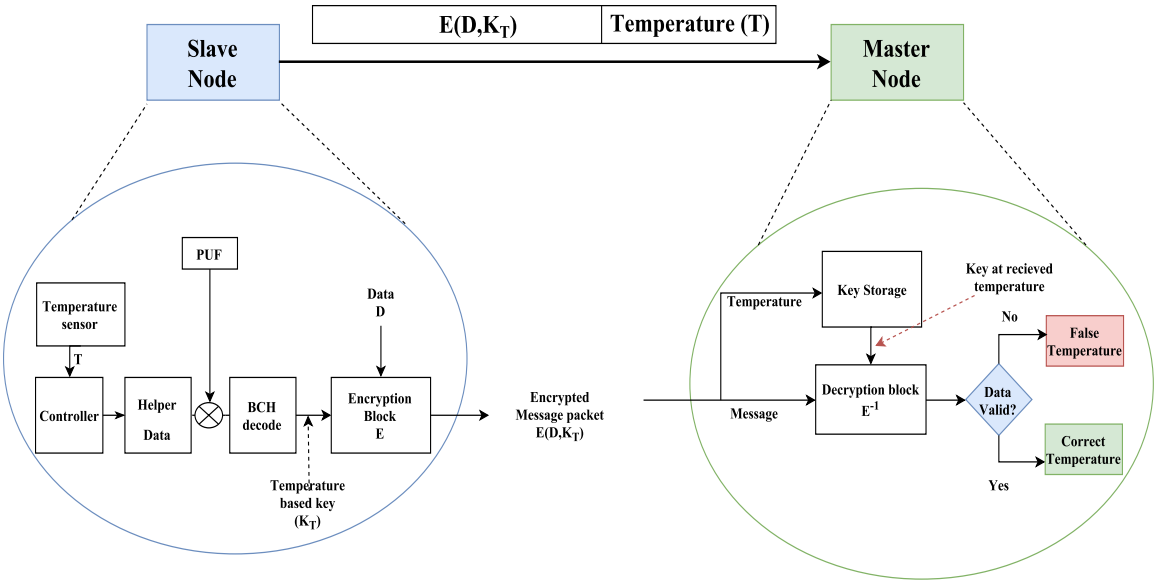


Figure 4.10: Complete system

4.4 Conclusion

We have shown the feasibility of using the SRAM PUFs for temperature based key establishment. We have shown the results across various technologies of SRAMs. We have also proposed a system for error correction and key generation based on temperature. Future work will include investigating schemes to narrow down the bits of SRAM that can be used to distinguish temperature and reduce the response size needed for the SRAM.

CHAPTER 5

ALTERNATE IMPLEMENTATION OF ANDERSON'S PUF ON XILINX FPGA

FPGAs are used for an increasingly large number of applications which require security. Due to their volatile nature, SRAM-based FPGAs require security at multiple levels. Bitstream encryption is often used to protect the configuration bits which define application implementation. Additionally, secure encrypt/decrypt cores are often implemented as part of a user's design to allow for the confidential processing of application data. These cores require secret keys that are often customized on a per-device basis.

In this section, we discuss the implementation of Anderson's PUF on SLICEL cells on FPGAs. The goal of this implementation is to allow the placement of encryption and key generation block on a fixed footprint on the FPGA. Also, for this PUF we will utilise per-device selection of PUFs on the block to reduce the size of the error correction circuitry on the FPGA. This work is a continuation of work done in [42].

5.1 Motivation

In this section we discuss the motivation behind this work. There were two primary reasons for the development of an alternate architecture. The first reason being the tuning of Hamming weight of PUF response and second being the resource constraint. We will discuss them in detail in the following sections.

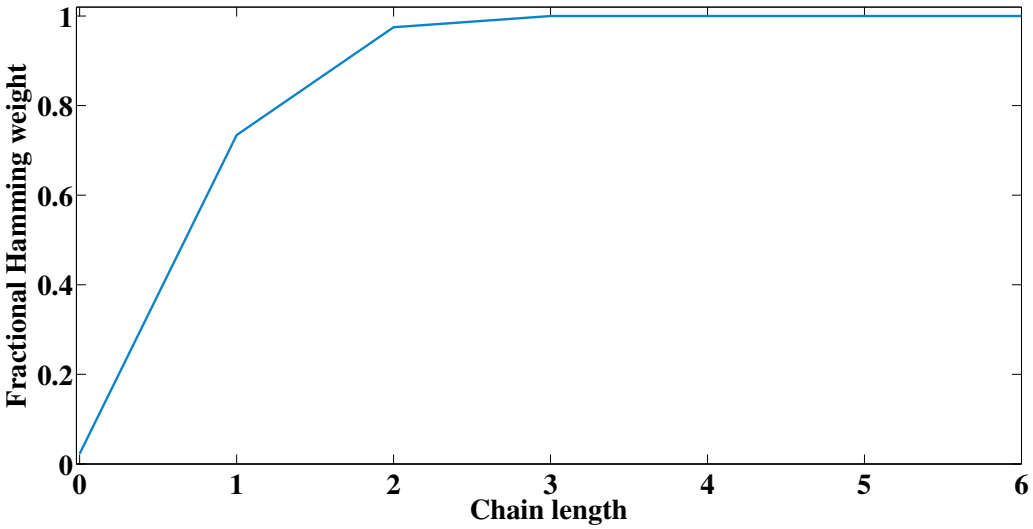


Figure 5.1: Figure showing the effect of elongating the carry chain on SLICEM based PUF's response.

5.1.1 Discrete stages for tune-ability

As discussed in section , the width of the glitch used to trigger the asynchronous set of the flip-flop depends on the number of stages in the carry chain. We can only change the number of stages in discrete steps. A problem arises when a device produce a balanced number of 0 and 1 responses for any length of carry chain. For Virtex-7 FPGAs we performed experiments to see the variation of Hamming weight with different number of stages. As is clear from fig. 5.1 there is no number of stages that balances the Hamming weight of the PUF response for the device.

With the absence of a point that balances the PUF response, the uniqueness of the PUF would be reduced thereby rendering it ineffective for key generation.

5.1.2 Design resource requirement

The design as originally proposed by Anderson can only be implemented on SLICEMs which are rare and fewer in number. The shift registers are used from SLICEM cells to ensure synchronous triggering of the Multiplexer select lines. This limits the placement of the PUFs on the device and also use up these limited resources

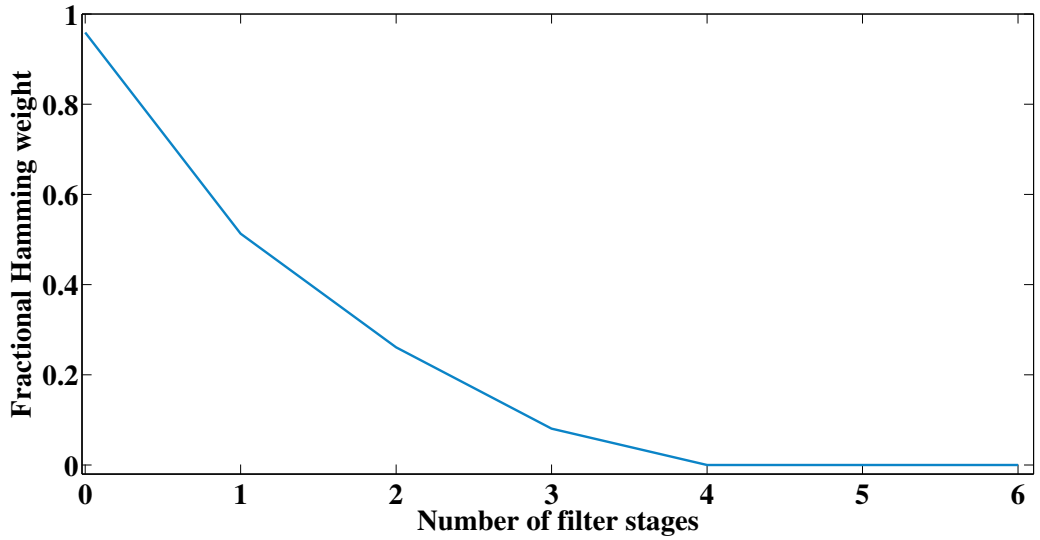


Figure 5.2: Figure showing the effect of adding Multiplexer stages in the chain. The glitch get filtered out, thereby narrowing it down and reducing the probability of setting the Flip-Flop.

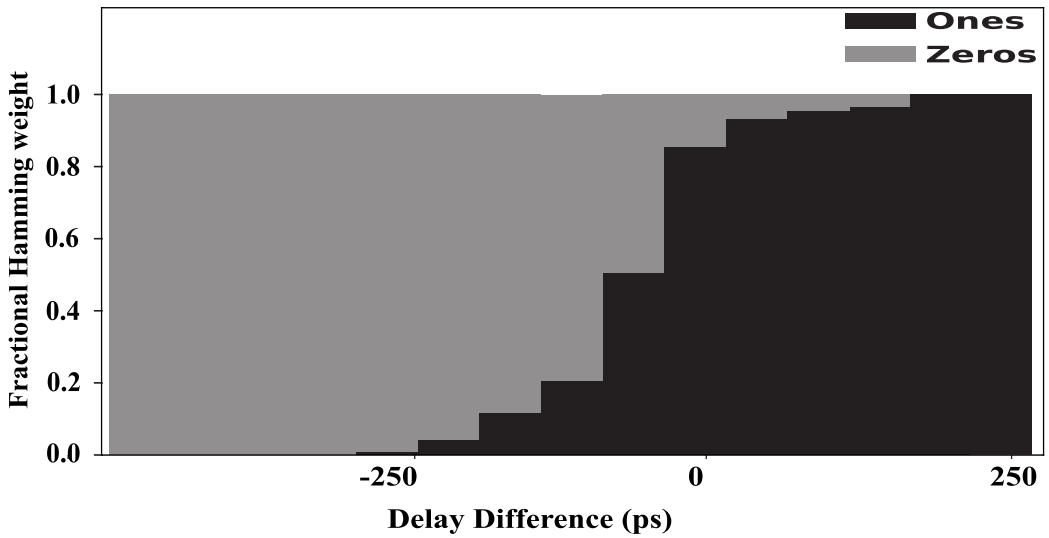


Figure 5.3: Figure showing variation of Hamming weight with change in delay difference between two paths for a carry chain of length 2.

which may be required for design purposes. The chip Xilinx Zynq-7020 contains 53,200 SLICEL cells and 17,400 SLICEM cells. We can clearly see that SLICEL cells are about 3 times the number of SLICEM cells in the chip. The original Anderson's PUF require the SLICEM cells. On the chip Xilinx Zynq-7020, there are 53,200 SLICEL cells and 17,400 SLICEM cells. We can clearly see that SLICEL cells are about 3 times the number of SLICEM cells in the chip. Therefore the block selected may not have adequate SLICEM cells to implement the PUF based key generation block.

5.2 The SLICEL PUF implementation and characterization

The novel implementation of Anderson's PUF that is proposed this study is shown in Fig. 5.4. It consists of two Xilinx SLICEs which can be either SLICEMs or SLICELs. We focus on SLICELs because previous works cannot make use of these. Relative to Anderson's PUF, our design replaces the shift register in each SLICEM with synchronous toggling signals created in each SLICEL from a flip-flop and a LUT configured as an inverter.

The timing operation of the design is shown in Fig. 5.5 and described here. Flip-flop $FF1$ in $SLICE2$ is initialized to a logic-1 and the corresponding flip-flop $FF1$ in $SLICE1$ is initialized to a logic-0. These flip flops will toggle their value in each cycle. A race condition that occurs after every second rising clock edge determines the width of the pulse that gets generated. The racing signals are as follows:

- On the clock edge, a rising transition propagates in SLICE2 from signal Q2 at the output of $FF1$ to signal $S2$ at the select input of multiplexer $M1$. The rising transition arrives when the 1-selected input of the multiplexer holds a logic-1 value, and causes the output of $M1$ to rise.

- On the same clock edge that triggered the above described sequence, in $SLICE1$ a falling transition propagates from signal $Q1$ at the output of $FF1$ to signal $S1$

at the select input of multiplexer $M1$ and then propagates upward through the delay chain. When this falling transition reaches the 1-selected input of $M1$ in $SLICE2$ it causes the output of the multiplexer to fall, terminating the pulse.

According to the race condition described above, the duration of the pulse as seen on signal $Pulse0$ can therefore be described by Eq. 5.1. Here, W_{glitch} is the duration of the pulse, D_{chain} is the delay through the length of the carry chain, D_{Q1-S1} is the delay from $Q1$ to signal $S1$ in $SLICE1$, and D_{Q2-S2} is the delay from $Q2$ to signal $S2$ in $SLICE2$.

$$W_{glitch} = D_{chain} + D_{Q1-S1} - D_{Q2-S2} \tag{5.1}$$

As shown in Fig. 5.4, the pulse on signal $pulse0$ can be propagated through three additional multiplexers in SLICE2, and this gives designers a choice about which pulse signal should be attached to the asynchronous preset signal of the flip-flop that will capture the PUF response. In the figure we have shown that $pulse0$ is the one attached to the flip-flop, but we will show in the next subsection how the choice among these pulses can be exploited to tune the Hamming weight of the PUF responses.

To optimize the PUF quality, we can use extra stages of multiplexer at the top of the selected carry chain to control the width of the glitch. Figure 5.2 shows the effect of adding extra multiplexer stages at the top of the carry chain. From the result we can see that adding 1 filter stage gives a Hamming weight close to the ideal value of 50%.

5.2.1 PUF Hamming weight tuning

To maximize the uniqueness of PUF responses, it is desirable to have the average Hamming weight of PUFs be close to 50%. In the Anderson PUF and our variation thereof, the average Hamming weight of the PUF response bits depends on the expected width of the glitch that arrives at the asynchronous preset input of the

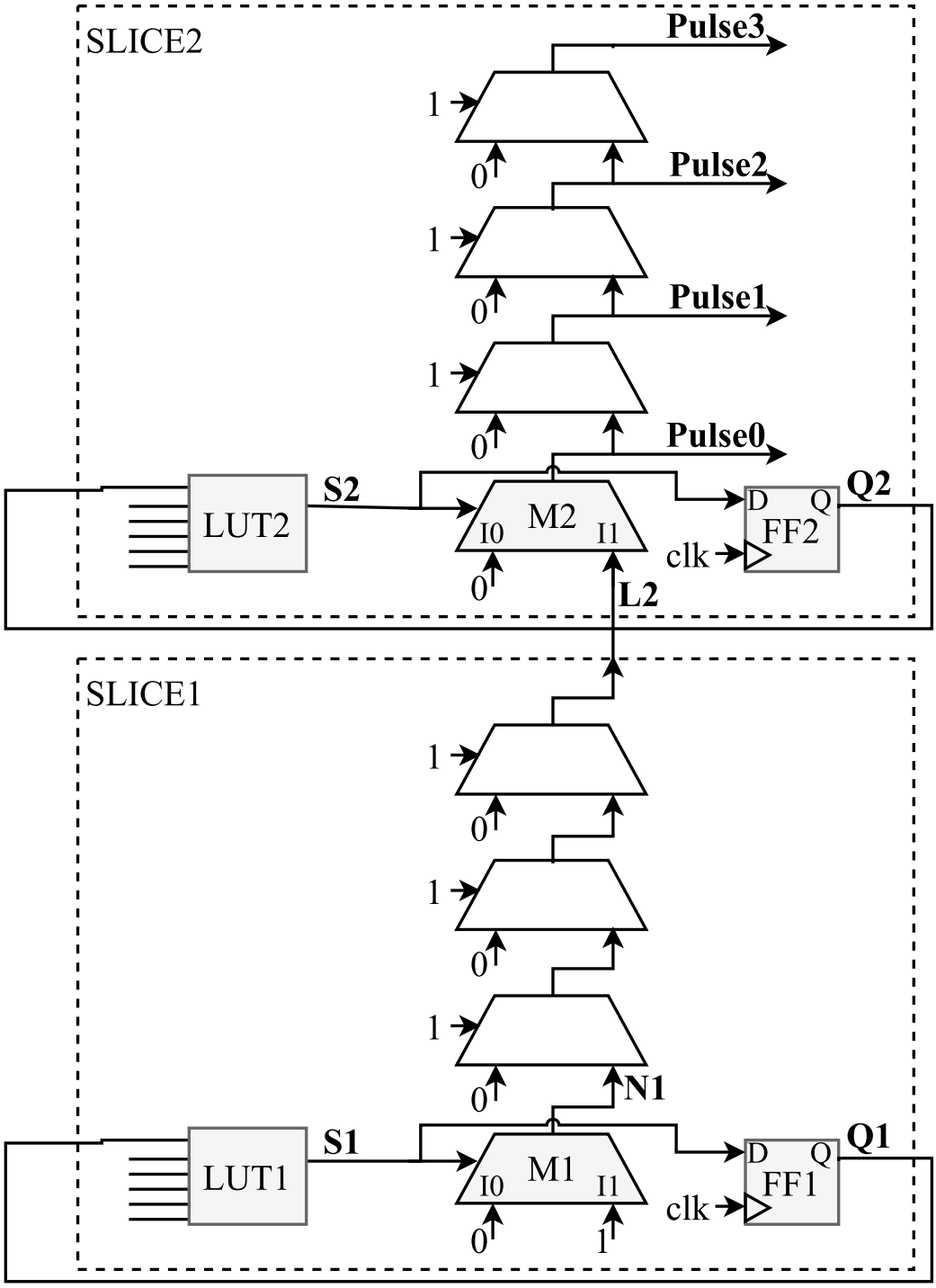


Figure 5.4: The alternate implementation of Anderson's PUF [3] implemented on a Virtex 7 architecture. The two LUTs are separated vertically by three Multiplexer stages and feed their corresponding flip-flops to generate a toggle signal for the select lines of their corresponding Muxes.

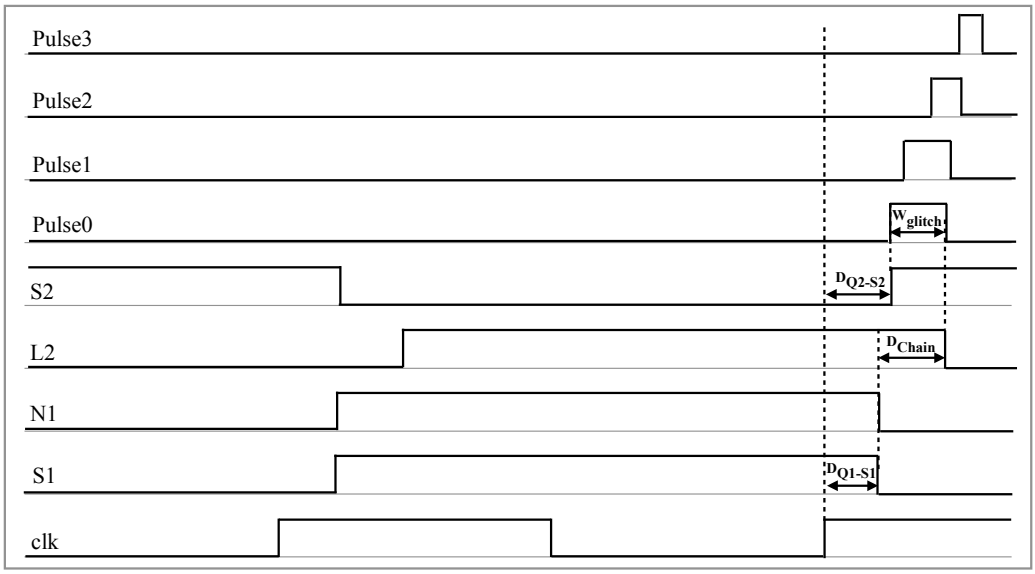


Figure 5.5: The timing waveforms shown govern the operation of glitch generation and glitch filtering in SLICEL PUF.

capturing flip-flop. A short glitch is less prone to setting the flip-flop value high, while a long glitch enhances the chance that the flip-flop will be triggered to store a response value of 1. Ideally, the average pulse pulse width will be on the cusp of the two response values, such that the response of each PUF instance will be caused by its process variations and their effect on the pulses of that PUF instance. We explore three different knobs that can be used to control the average Hamming weight of the PUF response bits.

Tuning by varying the chain length: Changing the number of delay stages in the design can be used adjust the term D_{chain} in Eq. 5.1, which describes the propagation time for a signal to propagate through the delay chain. As described by Eq. 5.1, changing this term has a direct impact on the glitch width, and therefore can be used to tune the Hamming weight of a design. The effect of changing the length is shown in figure 5.1. A longer delay chain increases the response Hamming weight.

Tuning by addition of filter stages: The second tuning parameter is the number of filtering stages used between the pulse generation and the preset input

of the flip-flop that generates the PUF response. To optimize the PUF response Hamming weight, we can use extra stages of the multiplexer at the top of the selected carry chain to control the width of the pulse. The addition of these filter stages attenuates the pulse before it reaches the preset of the Flip-Flop. Figure 5.2 shows the effect of adding extra multiplexer stages as filters. From the result, we can see that adding 1 filter stage gives a Hamming weight close to the ideal value of 50% when the delay chain comprises three multiplexers. Regardless of the delay chain length, adding more filter stages is found to reduce the Hamming weight.

Tuning by adjusting path delays: The third parameter to tune the Hamming weight of the PUF response can be done by varying the delay difference of the two paths that connect the output of the Flip-Flop to the input of the LUT. These delays are marked as D_{Q1-S1} and D_{Q2-S2} in figure 5.5. We can see that the these 2 delays affect how wide the glitch (shown as W_{glitch}) produced would be. Any increase in delay D_{Q2-S2} will delay the start of the pulse and thereby cause a corresponding reduction in glitch width. This effect of varying the delay difference is shown in figure 5.3. There is variation in Hamming weight of the response with the delay difference between these two paths.

To tune the path delays and change the 0-1 balance we begin with a PUF configuration having mux chain of length 2 with no filter stages. Random unconstrained routing is then performed on the feedback paths of the PUF by the Xilinx Vivado routing tool. Statistics are generated from the data acquired by evaluating the PUFs responses generated from the unconstrained routing to find what delay difference between the feedback routes produces the best balance in Hamming weight. The value of the delay difference that has the probability closest to 50 % for producing a one and a zero as the response is chosen as the desired value. In case of length 2 PUFs with no filter stages, we can see that a delay difference between -25ps and -75ps gives

the best balance (close to 50%). Once this value is known, we perform the following steps:

- For one PUF in placed in a block, we try different values on minimum delay constraints, in steps of 10ps, on each of the two feedback routes from the output of Flip-Flop to the input of LUT. This forces the routing tool to a find paths from the flip-flop to the LUT with a variety of different propagation delays.
- For each value of minimum delay, we record the obtained value after routing to generate a list of attainable delays and the specific routes that have these delays.
- From the two lists of delay values, one for each feedback path, we pick the values that produce the desired delay difference value which balances the response of the PUF.

This process is repeated one time for each PUF that resides in a different block because it has a different set of resources in the SLICE. The values generated are then used to generate proper delay and balance the response Hamming weight.

Our design uses a carry chain of length 3 with 1 filter stage and matched feedback paths. All the results presented hereafter in the paper will be with respect to this tuning.

5.2.2 PUF reliability & uniqueness

Reliability and uniqueness are the most important properties of a weak PUF. Reliability refers to the repeatability of PUF outputs over time, which lessens the burden of error correction. Uniqueness is a measure of how different the output values are across PUF instances. We quantify these two properties for the Virtex 7 architecture by analyzing the Hamming distances between pairings of 128-bit PUFs

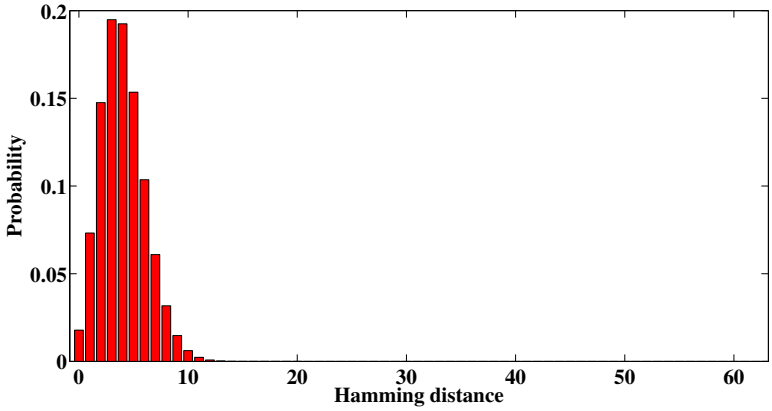
across ten chip instances. In total, we implement the 33 disjoint 128-bit PUFs on each chip and record 1000 output trials from each.

For reliability, we consider the distribution of within-class Hamming distances. Within-class Hamming distances are between outputs of two different trials from the same 128-bit PUF on the same chip. The histogram of Fig. 5.6a shows the distribution of Hamming distances for 63 million comparisons, representing all combinations of the 1000 trials that collected for each of the 33 different 128-bit PUFs on each of the 10 chips. The mean within-class distance is 4.0264 bits (3.14%).

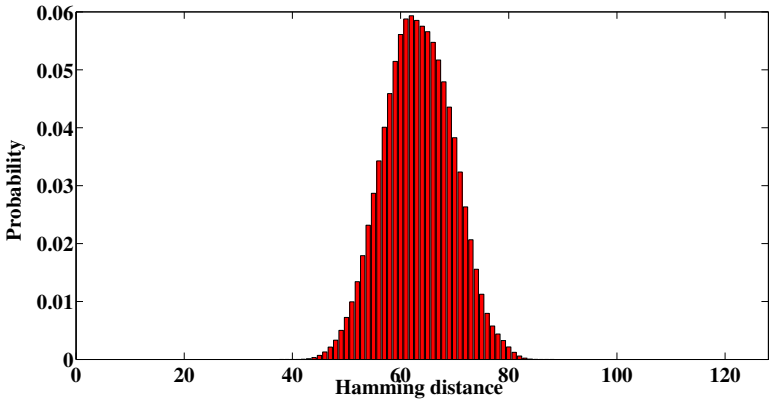
To study uniqueness we consider two different variants of between-class Hamming distance in our analysis. The first is the Hamming distance between outputs from two different randomly selected 128-bit PUFs that are on different chips or different locations on the same chip. Over 2.3 billion comparisons the mean distance is found to be 63.25 bits (49.41%) (Fig. 5.6b). The second set of between-class distances are similar to the first but confined to only compare PUF pairings that occupy the same locations on different chips; this case is interesting because it could show a reduced Hamming distance if PUF output values were significantly influenced by deterministic bias instead of device-specific process variations. Fig. 5.6c shows that, based on 660 million comparisons, the mean distance is found to be 63.12 bits (49.31%). The between-class results indicate that the PUFs produce highly unique outputs, and that the outputs are not being caused by deterministic bias associated with the location of the PUF on the chip.

5.2.3 Spatial Autocorrelation of PUF Location BERs

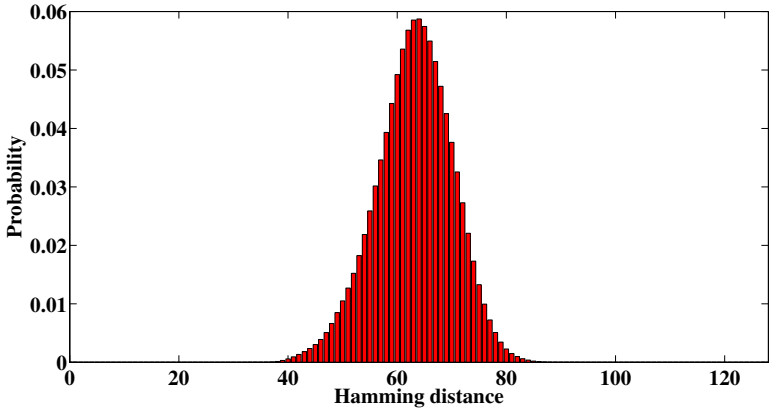
It is important to consider whether the PUF BERs are correlated spatially within each chip, as spatial correlation could imply a common cause for unreliability, instead of random per-device variations. The heatmap of Fig. 5.8 shows, for a single chip, the reliability of 4250 PUF instances according to their locations on the chip. Informally,



(a) Within class Hamming Distance



(b) Between class, all locations



(c) Between class, same location

Figure 5.6: Histograms of between-class and within-class Hamming distances of 128-bit PUFs. Within-class distances compare two measurements from the same 128-bit PUF instance. Between-class distances compare (in b) two different 128-bit PUFs on the same chip, or (in c) compare 128-bit PUFs that occupy the same locations on different chips. All measurements were made at room temperature of approximately 24^o and at the nominal supply voltage.

the lack of a clear pattern in this figure gives some visual indication that the unreliable PUFs are likely to be random and not highly clustered. To formalize the apparent lack of spatial correlation in Fig. 5.8a, we use Moran's I as a metric to quantify the spatial autocorrelation in the BER of PUF instances. For any single chip instance, Moran's I is computed using Eq. 5.3, where B_i and $\bar{B}$ are the BER of PUF instance i and the mean BER of the chip respectively. Computing Moran's I requires a spatial weight w_{ij} to indicate which PUF locations should be considered local to each other. For PUF locations i and j, we compute the weight w_{ij} as shown in Eq. 5.2, where r_i and c_i are row and column indices of the i^{th} PUF location. Restating this, the weight is set to 1 if the Euclidean distance between the row and column indices of two PUF locations is less than 10. Moran's I can take values between -1 and 1, where 1 indicates high spatial autocorrelation, and 0 indicates no spatial autocorrelation. The values of I obtained for all of the 10 chips are between 0.013 and 0.017, indicating that the unreliable PUFs do not tend to be highly clustered. An implication of the random position of unreliable PUFs is that it is not possible to simply choose certain positions for PUF placement that will be reliable across all chips.

$$w_{ij} = \begin{cases} 1 \text{ if } \sqrt{(r_i - r_j)^2 + (c_i - c_j)^2} < 10 \\ 0 \quad \text{otherwise} \end{cases} \tag{5.2}$$

$$I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(B_i - \bar{B})(B_j - \bar{B})}{\sum_i (B_i - \bar{B})^2} \tag{5.3}$$

5.3 Per-Device selection of PUFs

Given that our chosen design uses a delay path with 3 multiplexers and 1 stage of glitch filters to achieve a Hamming weight close to 50% (see Fig. 5.2), there are two options for placing the design in a pair of SLICEs. Fig. 5.7 shows the two possible configurations. The first two lines of Tab. 5.1 shows the between-class and within-class

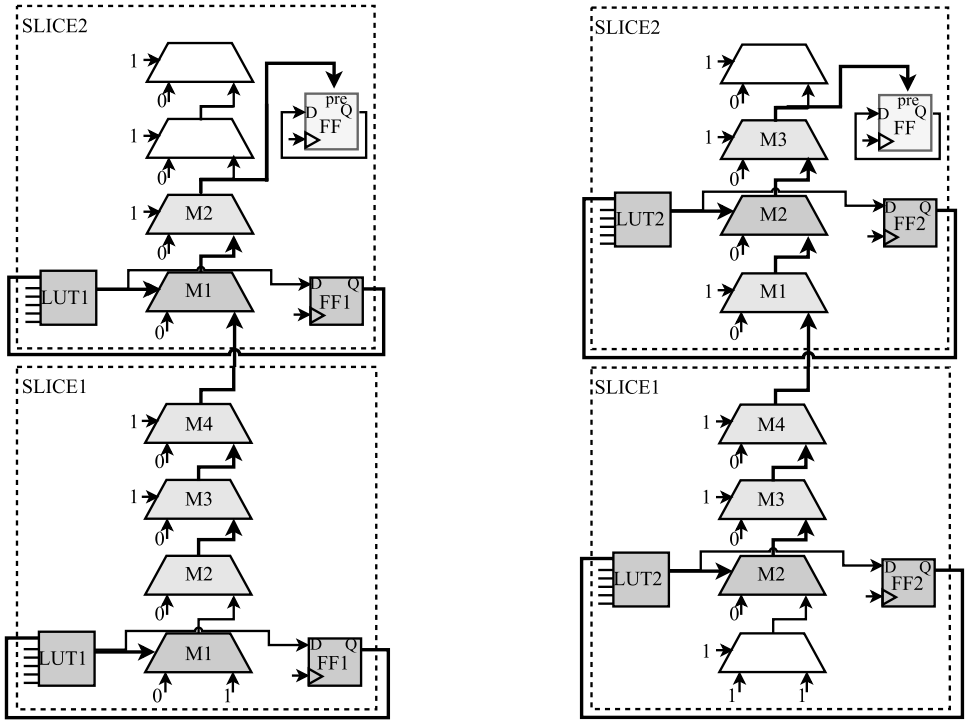
Table 5.1: Mean Hamming distance comparisons from 10 chips

Configuration	Within class	Between class
1	3.14%	49.31%
2	3.07%	46.09%

Hamming distances for each of these configurations. The low within-class Hamming distance shows good reliability in both configurations, and the close-to-ideal between-class Hamming distances show good uniqueness. This implies that two SLICEs can be configured in two different ways as a PUF, and in either configuration will have similarly desirable PUF statistics.

The average reliabilities of 3.14 and 3.07 are aggregated over many different PUFs with heterogeneous BERs. In reality, there are many PUF instances that are extremely reliable, and some that are very unreliable. At a single location of two slices, we will show it is often the case that one PUF configuration will be unreliable and the other will be reliable. Because of this, it is possible to create a highly reliable PUF by choosing the more reliable configuration in each of the PUF locations, but the choice must be made uniquely for each chip. The only scenario in which per-device configuration will not be beneficial is if both configurations for a given PUF bit are equally unreliable.

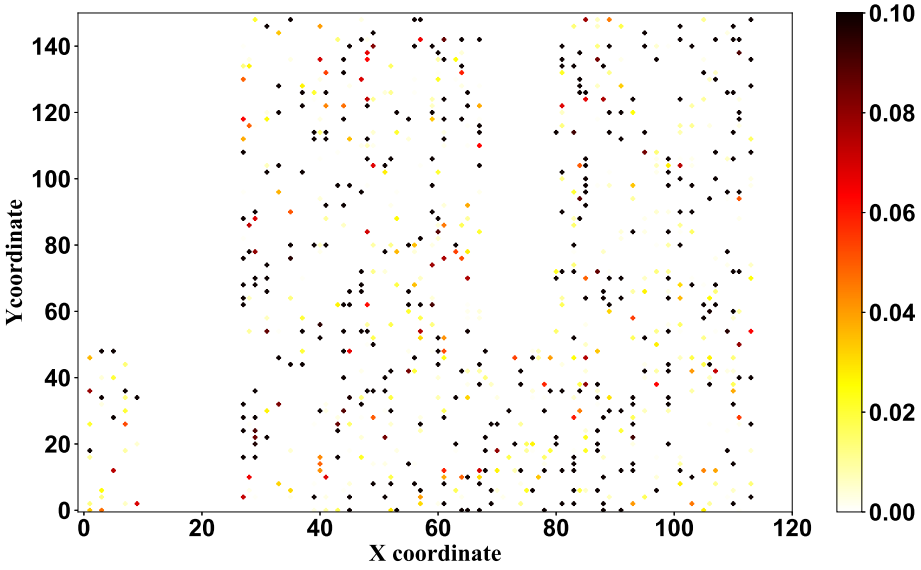
The benefit of performing best configuration selection is shown in table 5.1. The data is generated by taking the average of responses from 10 different chips. We can clearly see that the BER (within class Hamming distance) has gone down significantly without effecting the uniqueness of the PUF (between-class Hamming distance). This is also shown in figure 5.9. We can see that the fraction of reliable PUFs has increased. Reducing the BER of the PUFs has a great impact on the overall system cost in temrs area and power once error correction is considered.



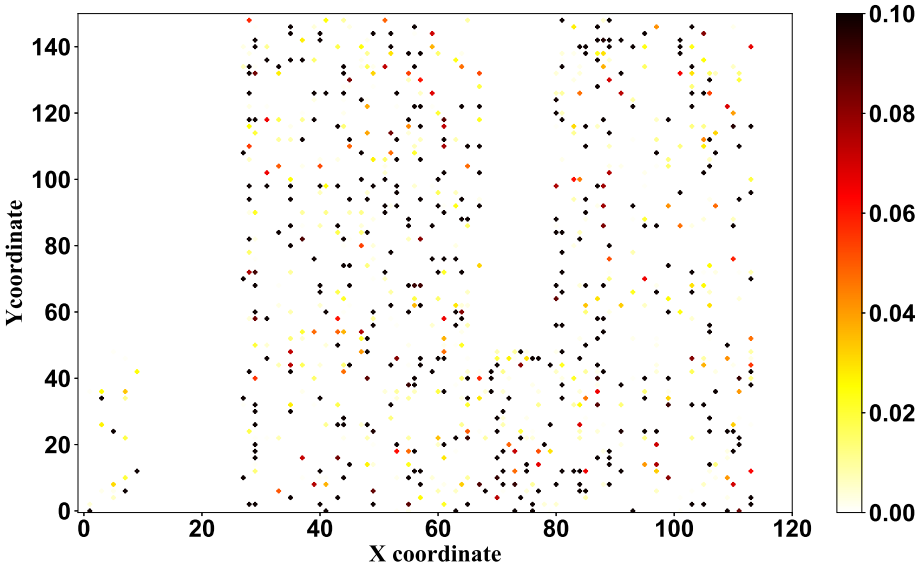
(a) configuration 1

(b) configuration 2

Figure 5.7: Two possible PUF configurations within the cell



(a) Heatmap of BERs of PUFs in configuration 1



(b) Heatmap of BERs of PUFs in configuration 2

Figure 5.8: Figure shows the BER of PUF instances placed at different locations on a chip. Unreliable instances are scattered and not concentrated in a particular area of the chip and uncorrelated across configurations.

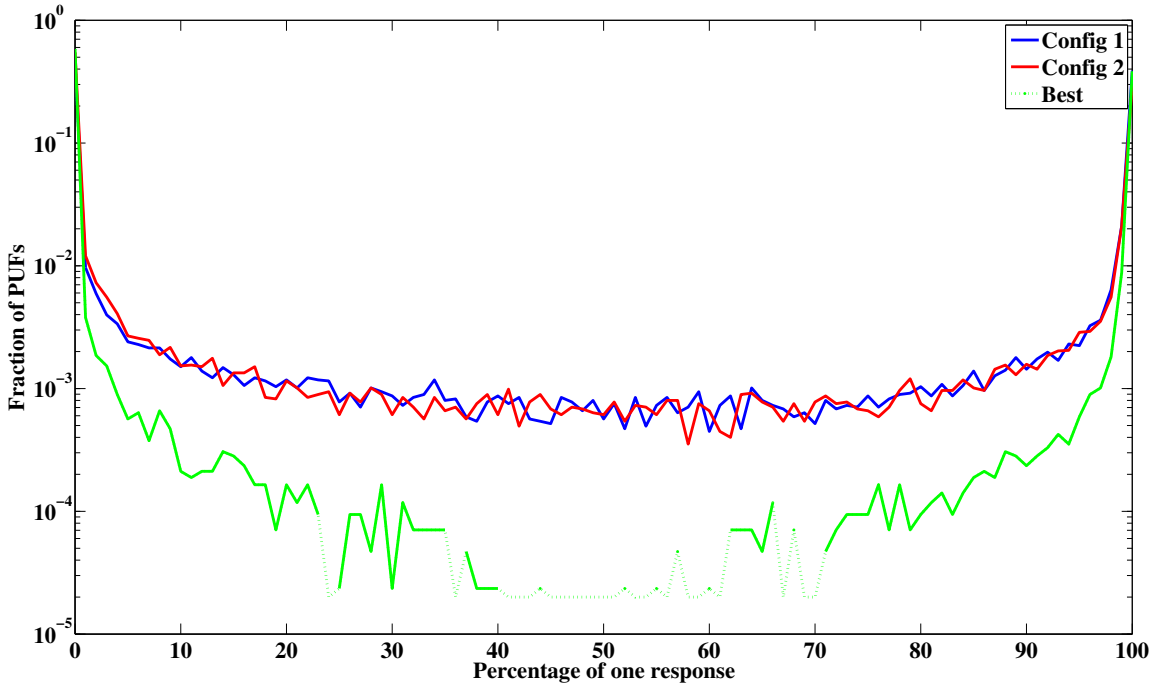
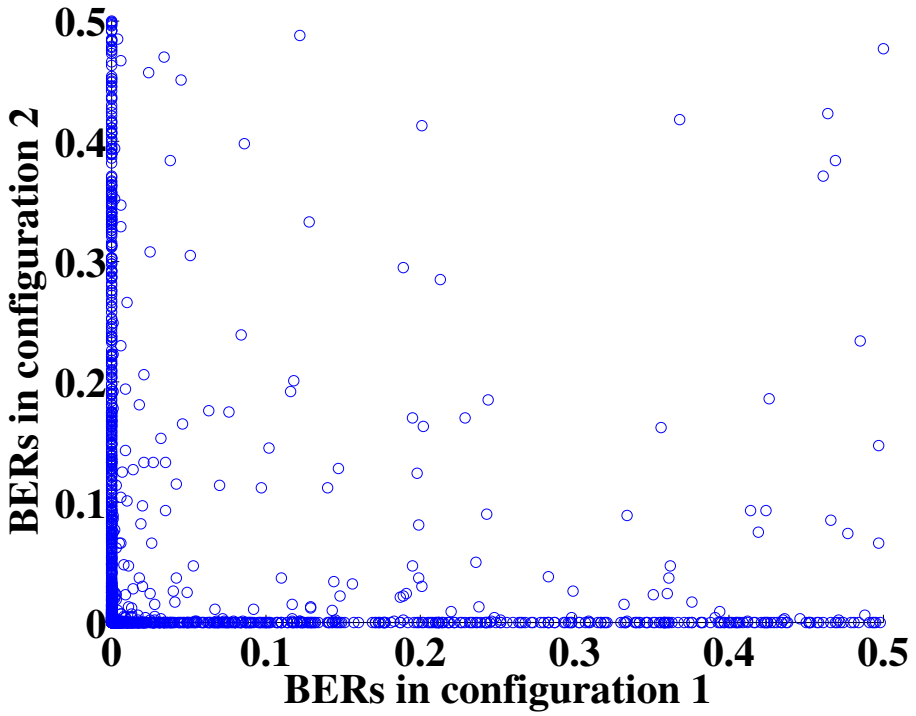


Figure 5.9: Figure showing the distribution of a 1 response of the PUF across 10 chips. A high probability closer to 0 or 100 percent implies that PUFs are highly reliable. Broken line indicates points where the value is zero.

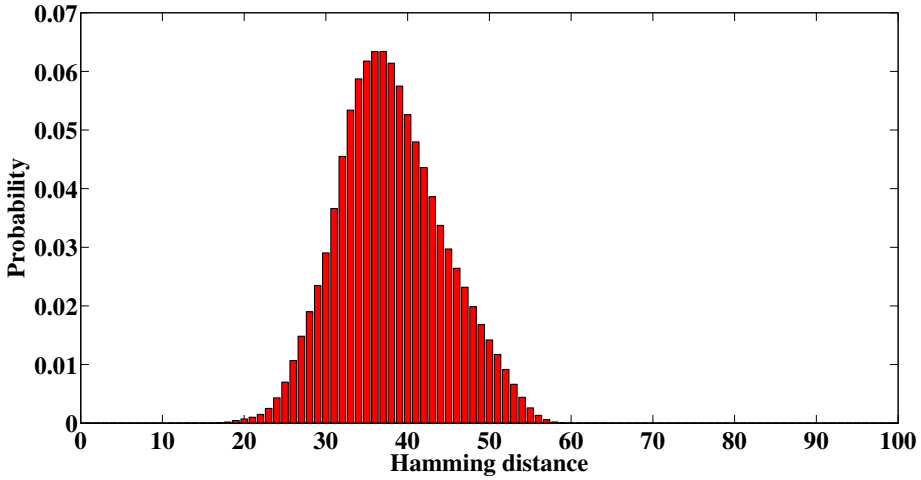
5.3.1 Correlation of between-configuration PUF responses

Either of the PUF configurations shown in Fig. 5.7 can be used in any two SLICELs, and as discussed in Sec. 5.3 we exploit this choice in order to choose the specific configuration that is found to be most reliable for each PUF location on a given chip. Noting that the two configurations share in common much of the delay chain, it is reasonable to wonder whether the two configurations would tend have similar reliabilities, which would eliminate the benefits of per-device configuration. Our findings in Fig. 5.10a show the BERs of the two different configurations do not tend to be correlated; if one configuration happens to have a high-BER PUF, it is not the case that the other configuration would also have a high-BER PUF.

In fact, despite the two configurations sharing a number of delay stages, the responses of the two configurations are relatively unique to each other, as shown in Fig. 5.10b. The fractional Hamming distance in this case is 38.02-bits (29.71%),



(a) BER in two different configurations



(b) HD between the configurations

Figure 5.10: When the same logic slices are configured in two different ways (see Fig. 5.7) on the same chips: (a) their BERs are uncorrelated; and (b) the fractional Hamming distance between responses from each configuration is 38.02-bits (29.71%).

which is less than ideal, but roughly ten times higher than the within-class distances. The uniqueness of the responses implies a large variation in the pulse widths generated in each configuration. Given that the width of each pulse is described by Eq 5.1, the finding implies that most the variability in the pulse width is coming from the random delay differences between the logically identical feedback paths that are unique to each configuration, and not due to the variations in the delay chain that are common to both configurations.

5.3.2 Design flow for per-device PUF configuration

Although selection of PUF configurations will require some amount of work to be done for each chip, it is infeasible for the entire design to be modified for each chip. We propose a CAD flow in which a global design is created that contains PUFs, and this design is then customized with a chip-specific partial bitstream that customizes its PUF configurations without going through the full place-and-route. The procedure uses the steps as described below:

- When performing place and route of the overall design, represent each PUF with a placeholder that uses the union of configuration 1 and configuration 2 resources. Later, for each instance, the placeholder will be replaced by a PUF in one configuration or the other.

- Create a bitstream that instantiates configuration-1 PUFs at each PUF location, along with testbench logic to collect many responses from each PUF instance and communicate back the results. Create a similar bitstream for configuration-2 PUFs.

- To deploy the PUF-containing design on a chip instance, do the following:

  - Apply the configuration-1 bitstream and collect results

  - Apply the configuration-2 bitstream and collect results

– For each PUF location, decide whether configuration 1 or configuration 2 has a lower BER.

– Customize the design bitstream by replacing the PUF placeholders with the specific choice of which PUFs to implement on that chip.

5.3.3 Temperature dependence of PUF response

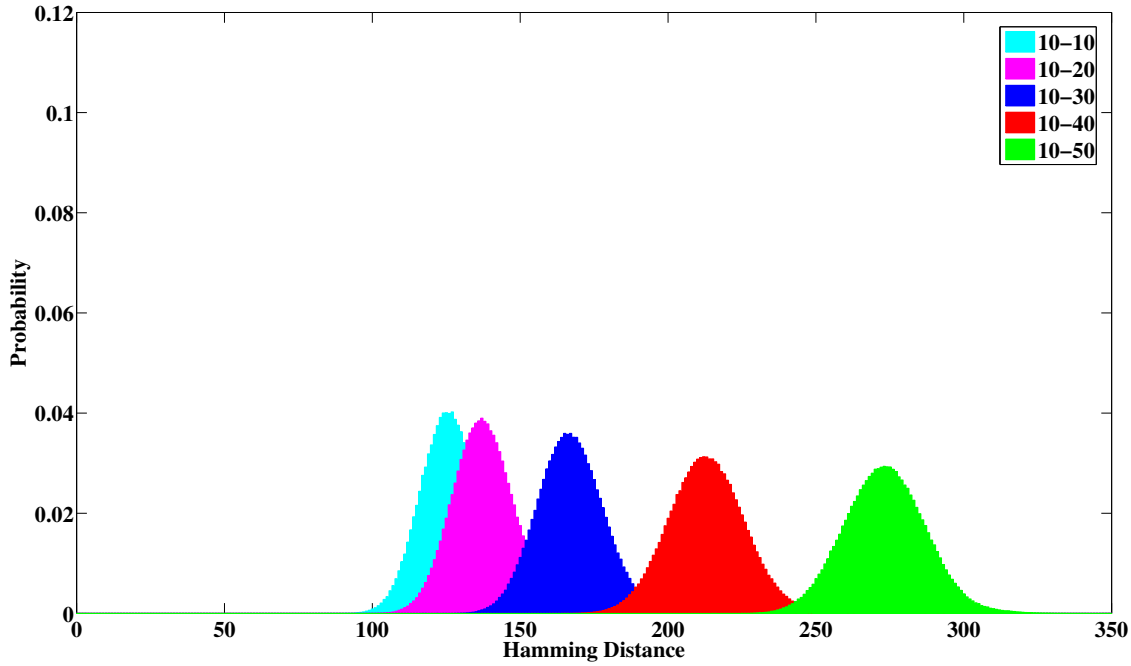


Figure 5.11: Effect of temperature on reliability of PUF(4250-bit) before best PUF configurations selection

In this section, we study the reliability of the proposed design with respect to temperature. The data was collected by putting the whole board in a TestEquity 115A Temperature Chamber. The soak time for each temperature was 10 minutes. After the soak, 1000 trials were collected from the chip and process was repeated for each temperature. Lastly, the on chip temperature sensor was used to verify the temperature after the soak time. The 10 degree Celsius trials were considered as the enrolled responses and comparisons were made with other temperature responses to

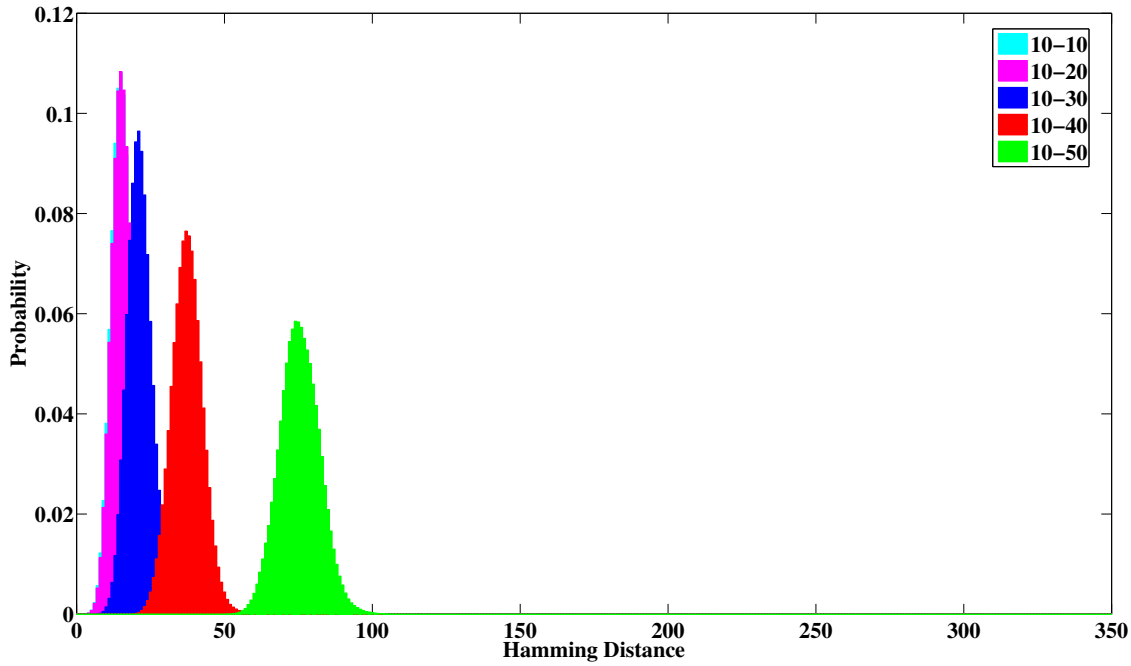


Figure 5.12: Effect of temperature on reliability of PUF(4250-bit) after best PUF configuration selection

see the variation in within class hamming distance with temperature. Fig. 5.11 shows the plot for five temperatures from 10°C to 50°C in steps of 10°C. Table 5.2 shows the average of BER for each of the five temperatures. The change in temperature causes an increase in the between-temperature Hamming distance, especially at higher temperatures.

We can increase the reliability of the design by selecting the best of PUFs on a chip from the two configurations. We select the best PUFs configuration for each PUF by generating a response at 20°C. The selection is made on the basis of BER of that PUF. The selected configuration for each PUF is implemented and then evaluated at all the temperatures from 10°C to 50°C. The data for reliability for the two configurations and after selection of best configuration is tabulated in Table 5.2 and plotted in Fig. 5.12. Even though the choice of which configuration to use is made based on

Table 5.2: Within class Hamming distance of PUFs (4250-bit) under different configurations at different temperatures

Temp(°C)	10	20	30	40	50
1	3.00%	3.26%	3.96%	5.04%	6.45%
2	2.86%	3.16%	3.94%	5.08%	6.60%
Best	0.39%	0.40%	0.53%	0.90%	1.80%

room temperature measurements, these PUFs are found to be more reliable across a range of temperatures.

CHAPTER 6

PUF BASED KEY GENERATION ON FPGAS

Keys used for cryptographic purposes must be repeatable to provide encryption and decryption of messages without error and to establish authenticity. To generate keys that are unique to each device we use PUFs. PUFs responses are noisy and tend to change over evaluations. This is not desireable as it can cause errors in key. To cope with this problem, fuzzy extractor [1][20][47] are used which uses a key along with the PUF data to generate a helper data. The helper data can be used later by the system during runtime to generate the key even with noisy PUF response. The key can be recovered as long as the PUF response error is within the correctable range of errors of the fuzzy extractor. We use a code-offset fuzzy extractor [1] construction with BCH codes for error correction in this work. In BCH codes, each code is described by a tuple (n,k,t); parameter n is the block size, parameter k is the number of information bits, and parameter t is the number of correctable errors.

Figure 6.1 shows the enrollment process of the key generated from the PUF. To generate a K-bit key using a BCH(n,k,t) with n-bits of output, k-bits of input data and t correctable errors, we have to split the K-bits of key into blocks of k-bits. For each block of the key, a n-bit codeword C_i is produced by the BCH encoder which gets offset by a n-bit PUF block R_i. This generates the n-bit helper data H_i for each k-bit blocks of the key. This helper data is then stored in the system to generate the key at a later time.

Figure 6.2 shows the key generation process. The Helper data H_i is prestored in the system and is used to generate the key. The PUF blocks are evaluated, which

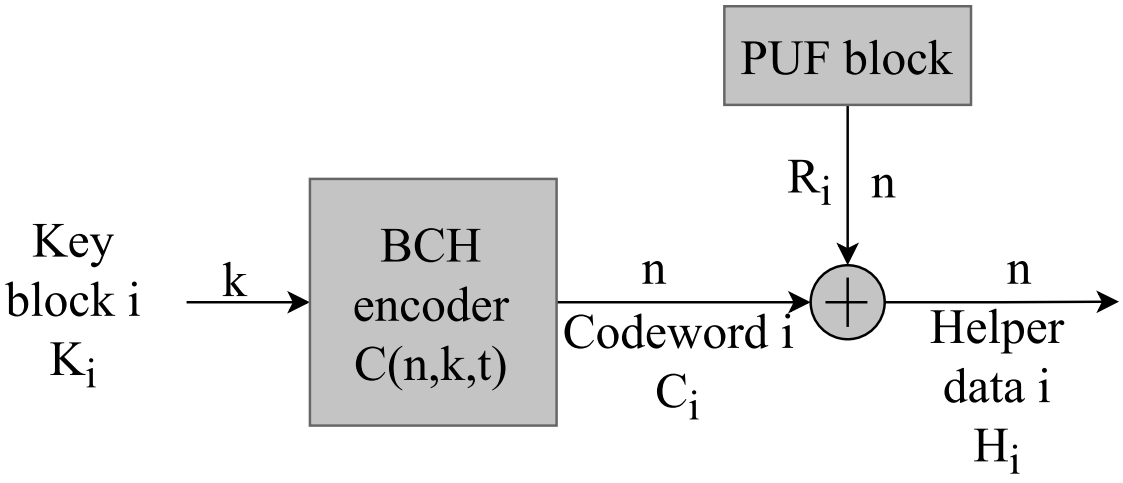


Figure 6.1: Block diagram showing the one time enrollment process for key generation

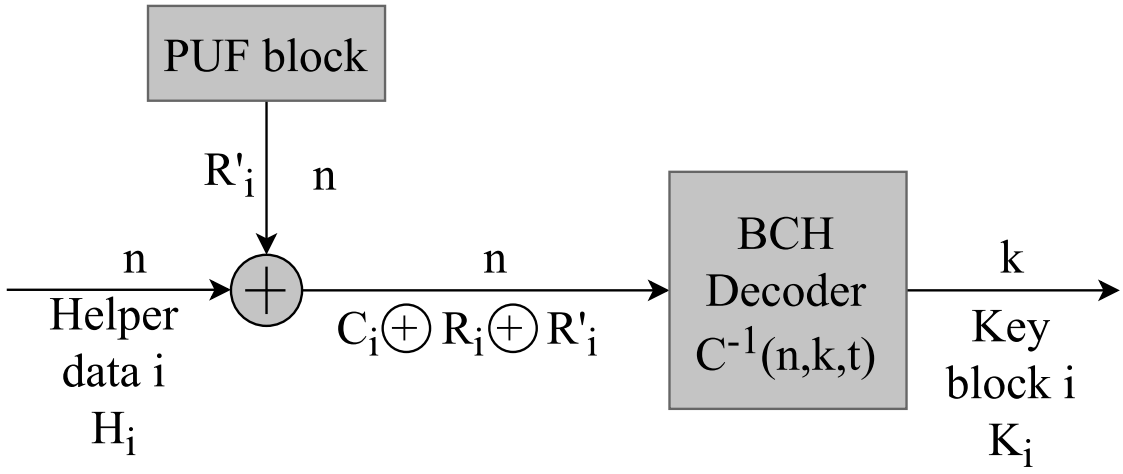


Figure 6.2: Block diagram showing the one time enrollment process for key generation

produce a response R_i' and offsets the corresponding helper data block H_i. The PUF output generated may slightly differ from the response R_i used during enrollment. The result is an invalid codeword C_i which is offset by the error bits in the PUF response. This invalid codeword can still be used to produce the original k-bit block of key as long as the number of errors in the PUF response were under t-bits. Thus the whole key can be generated as long as the PUF error doesn't exceed the maximum correctable errors of the BCH decoder.

6.1 Cost of Error correction

The cost of correcting the errors increases with the number of correctable errors. For the BCH decoder of block size n-bit, as the number of correctable errors t increases, the number of data bits k decreases. This results in an increase in the number of PUFs needed to offset the BCH encoded codewords. The complexity of the BCH decoder itself also increases. This results in a penalty in both area and power on the chip. For example, for a 255-bit block size, there can be two different codes used. One could correct 7 errors and carry 199 information bits while another could carry only 123 information bits but correct up to 18 errors. So, a 256-bit key generated from the above two codes would require 2 and 3 blocks respectively.

In traditional analysis, the bit error rate p_{bit} is calculated by averaging the error rates of all the PUFs and using it to generate the block failure rate. The probability of incorrectly decoding a block as p_{block}. This is computed using Equation 6.1 which shows the probability of finding more than t erroneous bits among n codeword bits when the bit error rate is p_{bit}. The probability of generating an invalid 128-bit key is denoted by p_{key} and denoted by Equation 6.2.

$$p_{block} = \sum_{i=t+1}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p_{bit}^{i}(1 - p_{bit})^{n-i} \tag{6.1}$$

$$p_{key} = 1 - (1 - p_{block})^{[128/k]} \tag{6.2}$$

This analysis ignores that not all the PUFs are homogenous and using an average bit error can result in an incorrect approximation of error. This would lead to a wastage in the area resulting from the increased size of error correcting circuitry.

6.2 A statistical model for PUF error correction

As discussed in previous section, the traditional model uses an average value of the BER to decide the size of error correcting code. This can be an under approximation

is there are some highly reliable PUFs or an over approximation if there are some highly unreliable PUFs. Both the cases are undesirable as the former will cause key failures and the latter will waste area and power on the chip. To cope with this problem, an accurate model that closely fits the statistics of a PUF is required to better predict the number of errors to be corrected [27].

6.2.1 Two parameter model

The response of the PUF is dependent upon two factors. The first one is the process variation in the silicon during manufacturing and the second one being the noise present at the time of response evaluation. A probabilistic PUF model is developed in [27] which takes into consideration the behavior of each individual PUF. It considers the probability of error for each individual PUF.

For the PUF i, the following variables are the observable variables:

- The One-probability(p_i) of a PUF instance i is the probability that the response of the PUF is a '1'upon an evaluation.
- The Error-probability($p_{e,i}$) of a cell producing a response different from the one generated at enrollment (golden response).

The PUF response characteristics are modelled using the following hdden variables:

1. Manufacturing process variation(m_i): This variable quantifies the amount of variation from the ideal design value and is determined once after the fabrication of the PUF and can be assumed to be constant throughout the lifetime of the PUF.
2. Noise variable(n_i^n): This variable quantifies the amount of noise that effects the response of the PUF at a every evaluation. The noise variable re-sampled at every evaluation of the PUF instance.

6.2.2 PUF model generation

To begin the modelling of the PUF behavior we first calculate the cumulative distribution function for one probability of the PUF response. Equation 6.3 [27] a is used to represent the one probability cumulative distribution function.

$$cdf_p(x) = \phi(\lambda_1 \phi^{-1}(x) + \lambda_2) \tag{6.3}$$

Here, $\lambda_1 = \sigma_N/\sigma_M$ and $\lambda_2 = (t - \mu_M)/\sigma_M$. σ_N and σ_M are standard deviations in process variation and noise variables respectively and μ_M and μ_N are means of the process variation and noise variables respectively. t is the threshold parameter which determines the response of the PUF. If the combined effect of noise and process variable is greater than t, the response will be a 1, otherwise the response will be a 0.

We use curve fitting tool in MATLAB to perform non-linear fit over variables (λ_1, λ_2) to minimize the mean square error of Equation 6.3 and the actual PUF data generated from the chip. A fit for configuration 1 PUFs is obtained at $\lambda_1 = 0.0711$ and $\lambda_2 = 0.1834$. Figure 6.3 shows that the model yields a close fit to the PUF statistics. We do the fitting for both the configurations and after performing best PUF selection.

By using the values of λ_1 and λ_2 in Equation 6.3, a one probability distribution can be obtained. This distribution can be used to generate generate PUF bit errors that actually resemble the values acquired from the chip.

$$pdfp_e(x) = \lambda_1(1 - x)\frac{\varphi(\lambda_1\phi^{-1} + \lambda_2) + \varphi(\lambda_1\phi^{-1}(x) - \lambda_2}{\varphi(\phi^{-1}(x))} \tag{6.4}$$

Equation 6.4 [27] gives the probability distribution function of the error probabilities where $\varphi(x)$ and $\phi^{-1}(x)$ refer to the probability density function of a normal distribution and the inverse of the cumulative distribution function of a normal distribution. To analyze the failure rates, we take 1000 samples of this distribution for values of x starting from 0 to 0.999 in steps of 0.001.

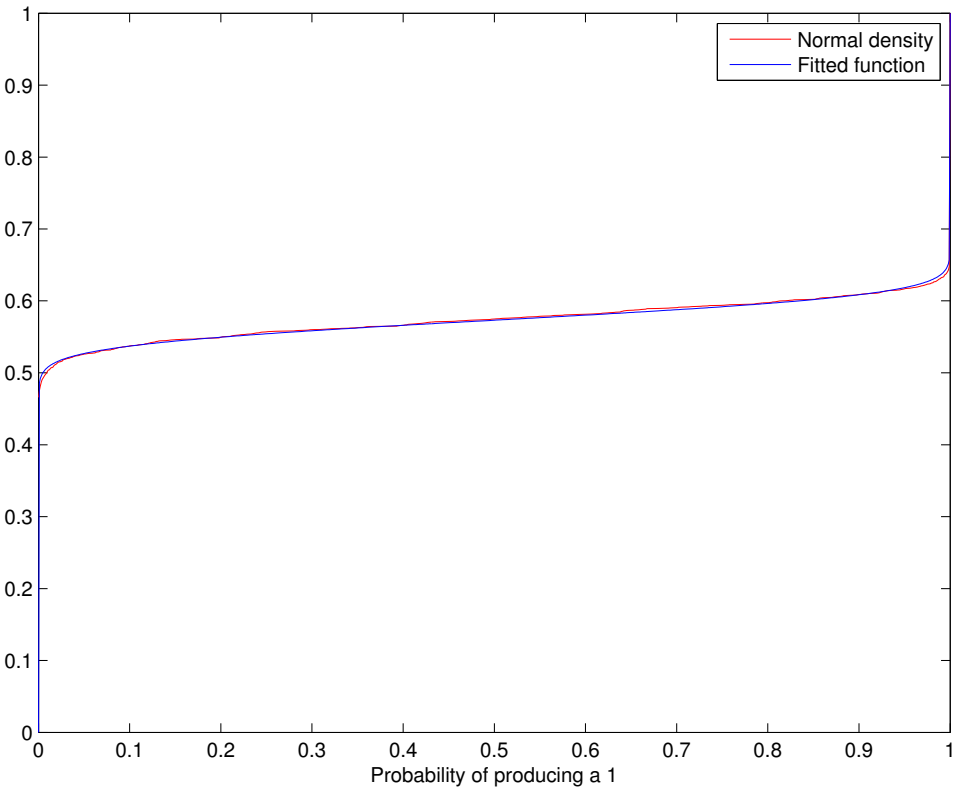


Figure 6.3: Fitted model vs actual data obtained at $\lambda_1 = 0.0711$ and $\lambda_2 = 0.1834$ for configuration 1

The key failure rate distribution is calculated for different error correcting codes. To calculate the block failure rate, we use the error probability model developed above. We use equation 6.5 to calculate the key failure rate. Here, n is the number of blocks of puf used to generate the key.

$$p_{fail} = 1 - (1 - p_{blockfail})^n \tag{6.5}$$

Probability of block failure is calculated using equation 6.6 [27]. In this equation, n is the number of pufs and t is the number of correctable errors. $F_{PB}(t; p_e^n)$ is a Poisson binomial distribution for the error rates of PUF and is given by 6.7. Instead of using fixed error rate, this distribution is used to generate more realistic block failure rates. Figures 6.4, 6.5 and 6.6 show the failure rate distribution for both the configurations

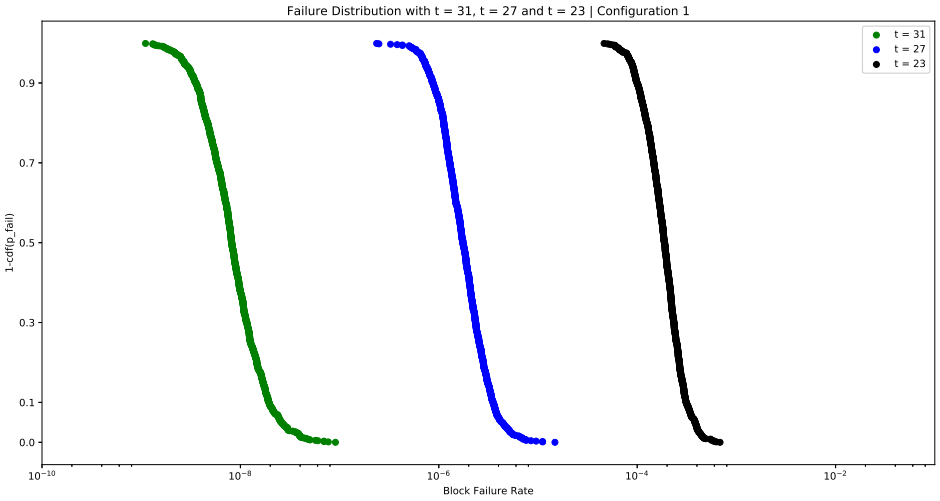


Figure 6.4: Key failure distribution with different number of error correcting BCH code for Configuration 1

and after selection of best configuration.

$$p_{blockfail}(p^n{}_e) = 1 - F_{PB}(t; p^n{}_e) \tag{6.6}$$

$$F_{PB}(t; p_e^n) = \frac{t+1}{n+1} + \frac{1}{n+1} \sum_{i=1}^{n} \frac{1 - C^{-i(t+1)}}{1 - C^{-i}} \prod_{k=1}^{n} (p_{e,k} C^i + (1 - p_{e,k})) \tag{6.7}$$

$$C = e^{\frac{j2\pi}{n+1}} \tag{6.8}$$

6.3 Results

We evaluate the error correcting needed to generate a key having maximum block failure rate of 10^{-6} for 99% of the chips using the PUF model created in section 6.2.1. In case of PUFs taken from the two configurations independently, from the data taken from 10 chips, the code needed to ensure the security metric described above is BCH(127,8,31). After performing the per-device selection of best PUFs from the two configurations we can go down to BCH(127,50,13) after evaluating PUF data from

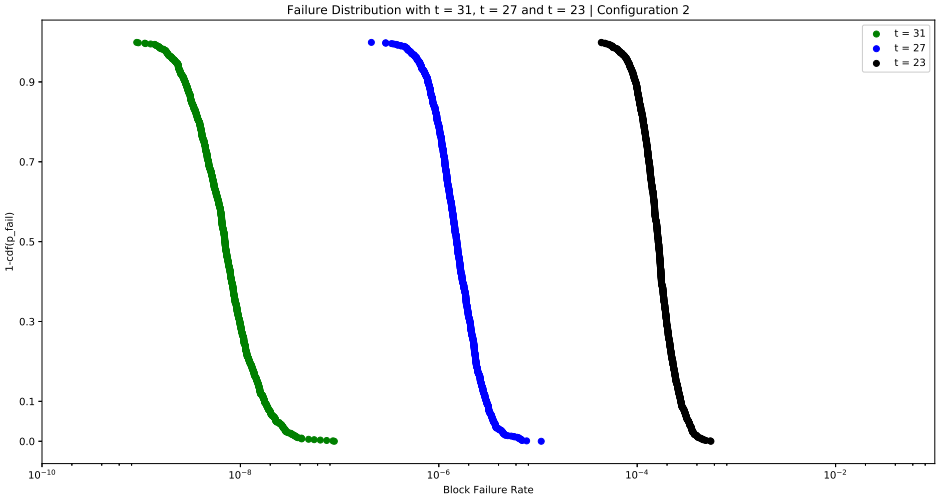


Figure 6.5: Key failure distribution with different number of error correcting BCH code for Configuration 2

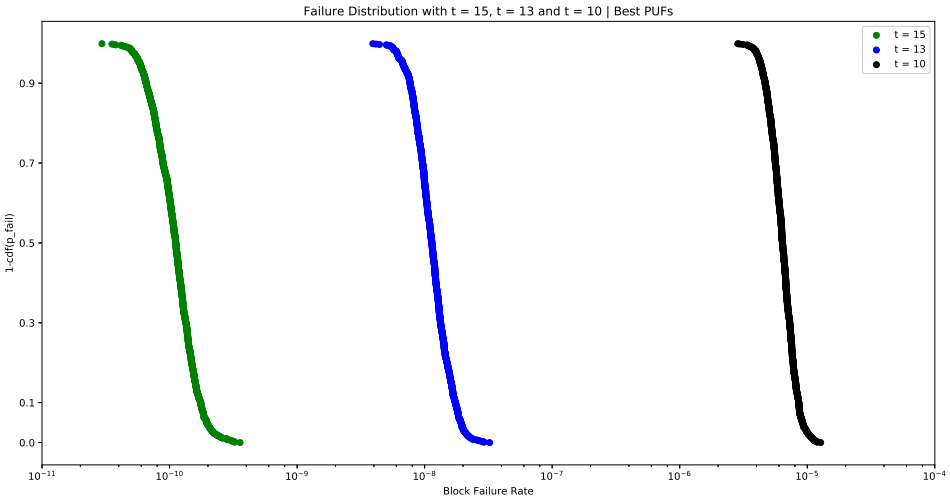


Figure 6.6: Key failure distribution with different number of error correcting BCH code for best configurations

Table 6.1: Resource requirements for 128-bit key generation

Mode	BCH code (n,k,t)	BCH Decoder		PUF		Total		Savings%	
		LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Best	127,50,13	1381	1088	762	1143	2143	2231	69%	72%
Normal	127,8,31	2904	1888	4064	6096	6968	7984		

10 chips. This clearly indicates that after performing best configuration selection the size of error correcting code circuitry has gone down significantly. Table 6.1 lists the different BCH codes and their resource requirements for implementation on a Xilinx Zynq-7020 chip. The area saving can be computed by comparing the requirements for the two cases. The total area savings are about 69% for LUTs and 72% for flip-flops.

CHAPTER 7

CONCLUSION

In this thesis, a scheme for temperature based keys using SRAM PUF is evaluated. SRAMs fabricated in different technologies have been evaluated and it is seen that this scheme is valid for all of them. We have shown that we can distinguish temperatures using PUFs. The resolution of temperature is dependent on the number of bits. The more the number of PUFs we have, the finer temperature difference we can distinguish.

A novel implementation of Anderson's PUF is developed for Xilinx FPGAs that can be instantiated anywhere on the chip. A scheme for increasing the PUF reliability using selection between PUF configurations is also shown. The effectiveness of the scheme is shown by calculating the error correcting code needed for a specified key failure rate using a statistical PUF model. It is evident from the results that there is a significant reduction in the area required by the error correcting code after doing per device selection of best PUFs.

BIBLIOGRAPHY

[1] Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. vol. 38, pp. 97–139.

[2] Alliance Memory Inc. *8K X 8 bit low power CMOS SRAM*, 2 2007. Rev. 1.

[3] Anderson, J. H. A puf design for secure fpga-based embedded systems. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Jan 2010), pp. 1–6.

[4] Chatterjee, U., Chakraborty, R. S., Mathew, J., and Pradhan, D. K. Memristor based arbiter puf: Cryptanalysis threat and its mitigation. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)* (Jan 2016), pp. 535–540.

[5] Chen, Q., Csaba, G., Lugli, P., Schlichtmann, U., and Rührmair, U. The bistable ring puf: A new architecture for strong physical unclonable functions. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust* (June 2011), pp. 134–141.

[6] Chen, Qingqing, Csaba, György, Lugli, Paolo, Schlichtmann, Ulf, and Rührmair, Ulrich. Characterization of the bistable ring puf. In *Proceedings of the Conference on Design, Automation and Test in Europe* (San Jose, CA, USA, 2012), DATE '12, EDA Consortium, pp. 1459–1462.

[7] Cherkaoui, A., Bossuet, L., and Marchand, C. Design, evaluation, and optimization of physical unclonable functions based on transient effect ring oscillators. Tech. Rep. 6, June 2016.

[8] Cypress semiconductors Inc. *64-Kbit (8K X 8) Static RAM*, 11 2014. Rev. G.

[9] Delvaux, J., Gu, D., Schellekens, D., and Verbauwhede, I. Helper data algorithms for puf-based key generation: Overview and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34*, 6 (June 2015), 889–902.

[10] Feiten, Linus, Spilla, Andreas, Sauer, Matthias, Schubert, Tobias, and Becker, Bernd. Analysis of ring oscillator pufs on 60nm fpgas. *European cooperation in science and technology*.

[11] Gassend, Blaise, Clarke, Dwaine, van Dijk, Marten, and Devadas, Srinivas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2002), CCS '02, ACM, pp. 148–160.

[12] Guajardo, Jorge, Kumar, Sandeep S., Schrijen, Geert-Jan, and Tuyls, Pim. *FPGA Intrinsic PUFs and Their Use for IP Protection.* CHES '07. Springer-Verlag, Berlin, Heidelberg, 2007, pp. 63–80.

[13] Halderman, J. Alex, Schoen, Seth D., Heninger, Nadia, Clarkson, William, Paul, William, Calandrino, Joseph A., Feldman, Ariel J., Appelbaum, Jacob, and Felten, Edward W. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM 52*, 5 (May 2009), 91–98.

[14] Holcomb, D. E., Burleson, W. P., and Fu, K. Initial sram state as a fingerprint and source of true random numbers for rfid tags. *Proceedings of the Conference on RFID Security 7*, 2 (01 2007).

[15] Holcomb, D. E., Burleson, W. P., and Fu, K. Power-up sram state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers 58*, 9 (Sept 2009), 1198–1210.

[16] Holcomb, Daniel E., and Fu, Kevin. Bitline puf: Building native challenge-response puf capability into any sram. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731* (New York, NY, USA, 2014), Springer-Verlag New York, Inc., pp. 510–526.

[17] Holcomb, Daniel E., Rahmati, Amir, Salajegheh, Mastooreh, Burleson, Wayne P., and Fu, Kevin. *DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification.* RFIDSec'12. Springer-Verlag, Berlin, Heidelberg, 2013, pp. 165–179.

[18] Hospodar, G., Maes, R., and Verbauwhede, I. Machine learning attacks on 65nm arbiter pufs: Accurate modeling poses strict bounds on usability. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)* (Dec 2012), pp. 37–42.

[19] Hwang, David D., Schaumont, Patrick, Tiri, Kris, and Verbauwhede, Ingrid. Securing embedded systems. *IEEE Security and Privacy 4*, 2 (Mar. 2006), 40–49.

[20] Juels, Ari, and Wattenberg, Martin. A fuzzy commitment scheme. In *Proceedings of the 6th ACM conference on Computer and communications security* (1999), ACM, pp. 28–36.

[21] Katzenbeisser, Stefan, Kocabaş, Ünal, Rožić, Vladimir, Sadeghi, Ahmad-Reza, Verbauwhede, Ingrid, and Wachsmann, Christian. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2012), CHES'12, Springer-Verlag, pp. 283–301.

[22] Kolberger, Andrea, Schaumüller-Bichl, Ingrid, and Deutschmann, Martin. *Risk Analysis of Physically Unclonable Functions*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 136–139.

[23] Kumar, A., Qin, H., Ishwar, P., Rabaey, J., and Ramchandran, K. Fundamental data retention limits in sram standby experimental results. In *9th International Symposium on Quality Electronic Design (isqed 2008)* (March 2008), pp. 92–97.

[24] Kumar, Sandeep S, Guajardo, Jorge, Maes, Roel, Schrijen, Geert-Jan, and Tuyls, Pim. The butterfly puf protecting ip on every fpga. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on* (2008), IEEE, pp. 67–70.

[25] Lofstrom, K., Daasch, W. R., and Taylor, D. Ic identification circuit using device mismatch. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)* (Feb 2000), pp. 372–373.

[26] Machida, T., Yamamoto, D., Iwamoto, M., and Sakiyama, K. A new mode of operation for arbiter puf to improve uniqueness on fpga. In *2014 Federated Conference on Computer Science and Information Systems* (Sept 2014), pp. 871–878.

[27] Maes, Roel. An accurate probabilistic reliability model for silicon pufs. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2013), CHES'13, Springer-Verlag, pp. 73–89.

[28] Maes, Roel. *Physically Unclonable Functions: Concept and Constructions*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 11–48.

[29] Maes, Roel, Van Herrewege, Anthony, and Verbauwhede, Ingrid. Pufky: A fully functional puf-based cryptographic key generator. In *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems* (Berlin, Heidelberg, 2012), CHES'12, Springer-Verlag, pp. 302–319.

[30] Maes, Roel, and Verbauwhede, Ingrid. *Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[31] Microchip Inc. *1Mbit SPI Serial SRAM with SDI and SQI Interface*, 1 2015. Rev. C.

[32] Rahmati, Amir, Salajegheh, Mastooreh, Holcomb, Dan, Sorber, Jacob, Burleson, Wayne P., and Fu, Kevin. Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks. In *Proceedings of the 21st USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 36–36.

[33] Rosenfeld, K., Gavas, E., and Karri, R. Sensor physical unclonable functions. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (June 2010), pp. 112–117.

[34] Rührmair, U., Martinez-Hurtado, J. L., Xu, X., Kraeh, C., Hilgers, C., Kononchuk, D., Finley, J. J., and Burleson, W. P. Virtual proofs of reality and their physical implementation. In *2015 IEEE Symposium on Security and Privacy* (May 2015), pp. 70–85.

[35] Rührmair, Ulrich, and Holcomb, Daniel E. Pufs at a glance. In *Proceedings of the Conference on Design, Automation & Test in Europe* (3001 Leuven, Belgium, Belgium, 2014), DATE '14, European Design and Automation Association, pp. 347:1–347:6.

[36] Rührmair, Ulrich, Sehnke, Frank, Sölter, Jan, Dror, Gideon, Devadas, Srinivas, and Schmidhuber, Jürgen. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 237–249.

[37] Rührmair, Ulrich, Sölter, Jan, Sehnke, Frank, Xu, Xiaolin, Mahmoud, Ahmed, Stoyanova, Vera, Dror, Gideon, Schmidhuber, Jürgen, Burleson, Wayne P., and Devadas, Srinivas. Puf modeling attacks on simulated and silicon data. *IEEE Transactions on Information Forensics and Security 8* (2013), 1876–1891.

[38] Sedra, Adel S., and Smith, Kenneth C. *Microelectronic circuits*, 7 ed. Oxford University Press, 2015.

[39] Spenke, Alexander, Breithaupt, Ralph, and Plaga, Rainer. An arbiter PUF secured by remote random reconfigurations of an FPGA. *CoRR abs/1610.04065* (2016).

[40] Su, Y., Holleman, J., and Otis, B. A 1.6pj/bit 96variations. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers* (Feb 2007), pp. 406–611.

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

[42] Vyas, S., Dumpala, N. K., Tessier, R., and Holcomb, D. E. Improving the efficiency of puf-based key generation in fpgas using variation-aware placement. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)* (Aug 2016), pp. 1–4.

[43] Willers, Oliver, Huth, Christopher, Guajardo, Jorge, and Seidel, Helmut. Mems gyroscopes as physical unclonable functions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 591–602.

[44] Xu, X., Rahmati, A., Holcomb, D. E., Fu, K., and Burleson, W. Reliable physical unclonable functions using data retention voltage of sram cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34*, 6 (June 2015), 903–914.

[45] Xu, Xiaolin, and Holcomb, Daniel. A clockless sequential puf with autonomous majority voting. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI* (New York, NY, USA, 2016), GLSVLSI '16, ACM, pp. 27–32.

[46] Xu, Xiaolin, Rührmair, Ulrich, Holcomb, Daniel E., and Burleson, Wayne. *Security Evaluation and Enhancement of Bistable Ring PUFs*. Springer International Publishing, Cham, 2015, pp. 3–16.

[47] Yu, M. D., and Devadas, S. Secure and robust error correction for physical unclonable functions. *IEEE Design Test of Computers 27*, 1 (Jan 2010), 48–65.

[48] Zeitouni, S., Oren, Y., Wachsmann, C., Koeberl, P., and Sadeghi, A. R. Remanence decay side-channel: The puf case. *IEEE Transactions on Information Forensics and Security 11*, 6 (June 2016), 1106–1116.