

Optimization of Physical Unclonable Function Protocols for Lightweight Processing

Carol Suman Pinto

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Patrick R. Schaumont, Chair

Michael S. Hsiao

Leyla Nazhandali

August 10, 2016

Blacksburg, Virginia

Keywords: Physical Unclonable Functions (PUFs), Static Random Access Memory
(SRAM), Cryptographic protocols

Copyright 2016, Carol Suman Pinto

Optimization of Physical Unclonable Function Protocols for Lightweight Processing

Carol Suman Pinto

(ABSTRACT)

Physically unclonable functions are increasingly used as security primitives for device identification and anti-counterfeiting. However, PUFs are associated with noise and bias which in turn affects its property of reliability and predictability. The noise is corrected using fuzzy extractors, but the helper data generated during the process may cause leakage in min-entropy due to the bias observed in the response. This thesis offers two optimization techniques for PUF based protocols. The first part talks about the construction of a secure enrollment solution for PUFs on a low-end resource-constrained device using a microcontroller and a secure networked architecture. The second part deals with the combined optimization of min-entropy and error-rate using symbol clustering techniques to improve the reliability of SRAM PUFs. The results indicate an increase in min-entropy without much effect on the error rate but at the expense of PUF size.

This research is supported by CISCO Systems Inc.

Acknowledgments

I would like to thank everyone who has played a crucial role in the pursuit of my Master's thesis:

Dr. Patrick Schaumont for taking me under his wing and providing guidance, knowledge, and encouragement during my work at Secure Embedded Systems Lab. I'm grateful for his belief and patience that drove this work to completion.

Dr. Michael Hsiao and Dr. Leyla Nazhandali for being on my review committee. Dr. Paul Plassmann for his advice throughout my academic journey here at Virginia Tech.

Dr. Aydin Aysu for his insights and assistance throughout my research at Secure Embedded System Lab and my colleagues Shravya Gaddam, Harsha Mandadi and Hemendra Rawat for their time and help.

My mother Jacintha Barboza, my aunt Precilla Katwa, my uncles Jossy Barboza, and Deven Katwa, without their love and support, I wouldn't be at Virginia Tech pursuing my Masters.

My friends and roommates who have been a constant source of support and encouragement.

Contents

Chapter 1	Introduction	1
1.1	Motivation	1
1.2	Contributions	3
1.3	Thesis Outline	4
Chapter 2	Background and Related Work	5
2.1	Physical Unclonable Functions	5
2.1.1	PUF Properties	6
2.1.2	Types of PUFs	7
2.1.3	Quality Measures	8
2.1.4	Cryptologic primitives	8
2.1.5	PUF Improvements	9
2.2	Secure Architectures	10

2.2.1	Basic Security Mechanisms and Concepts	10
2.2.2	Existing Security Solutions for Microcontrollers	11
2.2.3	SANCUS	12
2.3	PUF-based Protocols	14
Chapter 3 Design Optimizations: Processing component		15
3.1	Existing Model	15
3.1.1	System Architecture	16
3.2	Optimization using microcontrollers	19
3.3	PUF Enrollment Solution	19
3.3.1	Hardware architecture	19
3.3.2	Software architecture	21
3.3.3	Protocol	22
3.3.4	Security Analysis	22
Chapter 4 Design Optimizations: Coding Algorithms		24
4.1	SRAM Noise Pattern Analysis	24
4.1.1	Reason for noise in SRAM PUFs	24
4.1.2	Biased and Noisy PUF Model	25

4.1.3	Noise and probability distribution analysis using state flow diagrams	27
4.2	Symbol Encoding using Look Up Tables	32
4.2.1	Construction of Transition Tables	32
4.2.2	Typical grouping approaches	34
4.2.3	Grouping with a balance between probability distribution and error rate	36
4.2.4	Change in error correcting parameters	38
4.2.5	Security Analysis	38
Chapter 5	Experimental Results and Discussion	39
5.1	Enrollment Solution	39
5.1.1	Implementation Cost and Performance Analysis	39
5.2	Look up tables	41
5.2.1	Implementation	41
5.2.2	Performance Evaluation	41
Chapter 6	Conclusions and Future Work	52
6.1	Conclusions	52
6.2	Future Work	53
	Bibliography	55

List of Figures

2.1	A simple PUF representation	6
3.1	Device attestation using Fusion PUF [1]	17
3.2	Block Diagram of the Software Architecture of Fusion PUF [1]	18
3.3	Block Diagram of the Hardware Architecture of PUF Enrollment Solution . .	20
3.4	Block Diagram of the Software Architecture of PUF Enrollment Solution . .	21
4.1	Error rate and probability when SRAM bits of noise n and bias p are grouped into 2-bit symbols	26
4.2	Error rate and probability when SRAM bits of $n=0.1$ and $p=0.3$ are grouped into 2-bit symbols	28
4.3	Grouping based on probability	29
4.4	Grouping based on error rate	30
4.5	An alternate grouping option	31

4.6	State flow diagram for SRAM bits from a SASEBO board	33
5.1	Comparison of min-entropy values for SRAM for simulated p and n values .	43
5.2	Comparison of symbol error rate values for SRAM for simulated p and n values	43
5.3	Comparison of bit error rate values for SRAM for simulated p and n values .	44
5.4	Comparison of min-entropy values for SRAM on SASEBO board	44
5.5	Comparison of symbol error rate values for SRAM on SASEBO board	45
5.6	Comparison of bit error rate values for SRAM on SASEBO board	45
5.7	Comparison of min-entropy values for SRAM on DE2 board	46
5.8	Comparison of symbol error rate values for SRAM on DE2 board	46
5.9	Comparison of bit error rate values for SRAM on DE2 board	47
5.10	Comparison of min-entropy values for SRAM on PIC32 board	47
5.11	Comparison of symbol error rate values for SRAM on PIC32 board	48
5.12	Comparison of bit error rate values for SRAM on PIC32 board	48
5.13	Histogram of SRAM byte values of DE2 board upon 2 bit to 1 bit clustering	51
5.14	Histogram of SRAM byte values of SASEBO board upon 4 bit to 2 bit clustering	51

List of Tables

2.1	Comparison of Security solutions for Microcontrollers	13
4.1	Transition table for the SRAM data in fig. 4.1 where $p(00) = p * p$, $p(01) = p * (1 - p)$, $p(10) = (1 - p) * p$, $p(11) = (1 - p) * (1 - p)$	27
4.2	Transition table for the SRAM data in fig. 4.6	34
5.1	Hardware utilization in terms of logic cells	40
5.2	Software utilization in bytes	41
5.3	SRAM Datasets	42
5.4	Comparison of the three approaches in terms of BCH parameters for an expected entropy of 128 bits and FRR of 10^{-3}	50

Chapter 1

Introduction

In this chapter, we discuss the motivation and contributions of the thesis.

1.1 Motivation

Internet of Things has facilitated an increase in networked embedded systems in our everyday life. A severe problem faced by manufacturers these days is the illegal duplication of the remotely deployed networked devices. One method to avoid counterfeiting is to track carefully and control the production and deployment of these devices. This process can be done remotely using cryptographic techniques, including authentication. A typical method used to authenticate a device is using a secret key stored in a secure non-volatile memory region. However, invasive or side-channel attacks can result in the exposure of the secret key causing security failures. The above scenario led to the development of Physical Unclon-

able Functions (PUFs) which exploits the manufacturing variations of the device [2]. Since PUFs depend on the intrinsic characteristics of the device, it's hard to be manipulated or duplicated. Hence, PUFs form a more secure alternative to traditional secret key storage methods creating a potential solution to identifying authentic devices.

The first step of PUF-based authentication solutions is the enrollment where manufacturer extracts the PUF data to create a unique identifier for the device. Ideally, this step needs to take place in a secure execution environment. However, there isn't any work that provides a practical solution to the enrollment problem. The first part of the thesis tries to reduce the complexity of the processing element and the software overhead observed in a complete network integrated PUF-based solutions without sacrificing the security objectives.

SRAM PUFs [3, 4] are quite popular among all the PUFs. The reason is that the SRAMs are one of the standard components in most of the devices obviating the need to create additional hardware for the PUF. Also, the large size of SRAM offers a huge amount of data for analysis. The noisy bits observed due to environmental changes and aging need to be fixed to create reliable SRAM PUFs. A typical solution is to use an Error Correcting Code (ECC) to locate the noisy bits. But, implementing an error-correcting algorithm can turn out to be quite expensive in case of low-end devices. An efficient data processing algorithm for PUFs can not only improve reliability but can also enhance the min-entropy of the SRAM PUF-data, which is lost due to the bias in specific devices[5]. This change can improve the implementation costs for a resource-constrained device. The second part of the thesis analyzes the SRAM data from multiple devices and provides an efficient processing technique

that can eliminate the bias of the bits while keeping the error rate in consideration.

1.2 Contributions

This work offers the following resource optimization techniques for PUF based authentication protocols.

- Use of a lightweight Trusted Computing Base (TCB) to support enrollment.

The manufacturer needs to communicate securely with the device, attest the PUF data extraction software and collect the data in an isolated execution environment.

The above design technique serves as a solution for secure enrollment of PUF data provided isolation be achieved using appropriate secure architectures for low-end resource-constrained devices.

- Use of lookup tables to support combined optimization of bias and bit error rate.

The byte level dependencies observed in SRAM data [5] is exploited using look up tables by careful and simultaneous analysis of bias and bit error rates. This technique of using simple lookup tables provides a significant improvement in min-entropy and symbol error rate which can result in a small implementation of decoding logic.

1.3 Thesis Outline

In Chapter 2, we summarize the existing work on physical unclonable functions and secure architectures. Chapter 3 and Chapter 4 describes our design which reduces the complexity of the processing component, and the coding algorithm. Chapter 5 evaluates the performance of our implementation. Ultimately, Chapter 6 concludes this work.

Chapter 2

Background and Related Work

In this chapter, we discuss the background of two main elements of thesis - Physical Unclonable Functions and Secure Architectures.

2.1 Physical Unclonable Functions

Physically Unclonable Functions (PUFs) are physical objects which are unclonable across circuits due to random manufacturing variations of the underlying physical device [2]. These defects also impact the challenge-response behavior of the PUF.

For a given m-bit Challenge C_i , Response $R_i = \text{PUF}(C_i)$, where R_i is of n-bits. PUF serves as a function that provides a mapping between challenges and responses.

Figure 2.1: A simple PUF representation



2.1.1 PUF Properties

The following properties need to hold for a PUF:

Reliability

On querying the same challenge multiple times, a PUF will return a same response with high probability. The different responses are due to noise induced by the environmental variations such as ambient temperature, voltage ramp up curves, the aging process, etc. The stabilization is then achieved using a suitable error correction method.

Unpredictability

It is infeasible for an adversary to replicate physically the PUF such that it would provide the exact response for every possible challenge. Also, it's hard to guess the response to an unknown challenge, even if the adversary has observed multiple challenge-response pairs.

2.1.2 Types of PUFs

Memory-based and Delay-based PUFs

Maes et al. explain how the positive-feedback loops which are used to store the bits contribute to the device-specific characteristics of memory-based PUFs [6]. The perfectly balanced loop has an equal chance of becoming logic 1 or 0, but the manufacturing variations introduce an imbalance resulting in an unpredictable power-up state. SRAM PUFs [3, 4], Buskeeper PUFs [7], Latch PUFs [8] and D Flip-Flop PUFs [9] are the most widely used memory-based PUFs.

Delay-based PUFs exploit the race conditions within the integrated circuit [10]. They include Arbiter PUFs [11] and Ring Oscillator PUFs [12]. Arbiter PUF is based on the delay difference observed in two identical paths caused due to the process variations while Ring oscillator PUF is based on the random frequency variations on identical ring oscillators [6].

Strong and Weak PUFs

Strong PUFs exhibit an enormous number of challenge-response pairs. This vast number makes it impossible to clone physically with the exact behavior or to predict response to a random challenge. Also, the finite-read out speed ensures that a software learning model will never be able to access all the challenge-response pairs [13].

Weak PUFs support only a small number of challenge-response pairs. These responses can be used as a unique key or seed in encryption protocols [14].

2.1.3 Quality Measures

A PUF's reliability is evaluated using its average Intra-Hamming distance over the output responses to identical challenges on the same PUF instance. PUF's uniqueness is assessed using the average Inter-Hamming distance over the output responses to the identical challenges on different PUF instances [15].

2.1.4 Cryptologic primitives

The PUF responses need to be reliable and unpredictable for use in many PUF-based applications [16]. PUFs are usually combined with helper data algorithms to fix the noisy and the uniformly random behavior. Delvaux et al. [17] provide an overview of the helper data algorithms for PUF-based key generations.

Generation Phase and Reproduction Phase

During the generation phase, the PUF response of the device to a specific challenge is recorded and enrolled at a secure location. The reproduction phase involves observing the PUF response at a later time, locating the error bits and fixing them. This regenerated response is used to compare against the response observed during enrollment.

Fuzzy Extractor and Reverse Fuzzy Extractor

Fuzzy extractors [18] are used to correct the noisy bits to generate PUF responses useful for cryptologic operations. The algorithm assumes that the verifier sends the response and the error decoding algorithms in the reproduction phase execute at the verifier side. Van Herrewege et al. proposed Reverse Fuzzy Extractor [19] where the compute intensive error decoding takes place at the verifier side.

2.1.5 PUF Improvements

Reliability

The reliability issue of SRAM PUFs is solved using Error Correction Codes(ECC). But the information leaks and computational costs limit the use of ECC. Bhargava et al. proposed reliability enhancements for SRAM PUFs by accelerated aging, multiple evaluation and using activation signal controls [20]. Cortez et al. proposed matching of voltage ramp-up time to ambient temperature to improve the SRAM PUF reliability [21]. Bit selection algorithm [22] utilizes the information of reliable bits observed during enrollment for better results. Yu et al. proposed a Maximum Likelihood symbol coding method for PUF bits with bit error improvements [23].

Unpredictability

Several debiasing schemes are proposed that transform the biased PUF distributions into a non-biased PUF distribution resulting in a high min-entropy, but with the assumption that biased PUF response bits are i.i.d bits. Index-based syndrome coding [24] proposed by Yu et al. provides significant benefits over XOR masking by not leaking additional min-entropy. Maes et al. proposed von Neumann-like extractors for several variations of debiasing schemes to achieve better reliability of PUF bits [25].

2.2 Secure Architectures

The growing number of network connected devices pose a major security requirement to ensure that the system properties are not affected by the presence of an adversary. PUF-based device authentication solutions are deployed at a remote location, and they need to be running securely for the verifier to ensure that an adversary has no access to it.

2.2.1 Basic Security Mechanisms and Concepts

The conventional security mechanisms such as platform integrity, secure storage, isolated execution, device authentication, attestation, and provisioning employ trust anchors such as device keys, isolated memory or cryptologic mechanisms [26].

- Platform integrity is achieved when the manufacturer can detect or prevent the unau-

thorized modifications to its platform either at boot or runtime time.

- Secure storage enables denial of unauthorized access to securely stored data.
- Isolated execution is supported when a secure code can execute while being immune to malicious code in the environment.
- Device authentication is achieved by the verifier when it can identify the device.
- Attestation involves verifying the software running on the device while provisioning includes secure transmission of data to the trusted software on the device.
- Secure communication suggests that the data exchange between the verifier and the device happens in a protected manner.

2.2.2 Existing Security Solutions for Microcontrollers

Security mechanisms can be implemented using secure coprocessors [27], hardware-assisted virtualizations [28], Trusted Platform Modules (TPMs) [29], Trusted Software Modules (TSMs) [30], secure microkernels [31], and trusted hypervisors [32, 33]. These approaches are very expensive in terms of manufacturing cost and energy consumption in case of resource-constrained devices [26].

The basic security services for low-end devices can be implemented using purely software based attestations that depend on the response time [34, 35]. CARMA [36] reduces the hardware trusted computing base (TCB) by leveraging the Cache-as-RAM execution mode

and not trusting the peripherals, buses, and the memory. SMART [37] offers minimal hardware architecture for trusted execution by ensuring that the secret key is available only when executed from the trusted code present in ROM. Other approaches such as SANCUS [38] manage tasks using CPU instructions and implement Memory Access Logic(MAL) to avoid the attacks due to malicious code. However, any violation of the code in case of SMART [37] and SANCUS [38] result in the reset of the device. TrustLite [39] enables multiple protected tasks to run in parallel by providing a programmable, execution-aware memory protection (EA-MPU) subsystem. It also avoids the problem of reset using Secure Loaders which restore the necessary access protection rights. TYTAN [40] also makes use of the execution-aware memory protection (EA-MPU) while supporting secure inter-process communication, local and remote attestation. Table 2.1 provides a comparison of different security mechanisms among the selected few security approaches.

The recent device authentication and secure key generation applications use PUF based solutions. PUFs eliminate the requirement of storing the key in a secure nonvolatile memory, hence are immune to invasive attacks [41].

2.2.3 SANCUS

SANCUS [38] is a security architecture developed for low-end resource-constrained networked embedded devices. The security mechanisms are implemented as mentioned below.

- Isolated Execution: This is achieved by the use of a Memory Access Logic (MAL)

Table 2.1: Comparison of Security solutions for Microcontrollers

	CARMA	SANCUS	SMART	TYTAN
Platform Integrity	Keys generated by SAKE [42]	Key based	HMAC Correctness	Secure boot and EA-MPU*
Secure Storage	-	Memory Access Logic	Key access controls	Key associated with a task
Isolated Execution	Cache-as-RAM	Memory Access Logic	ROM execution control	Execution aware memory protection unit
Device Authentication	SAKE [42]	Key based	Key based	RTM* Measurements
Attestation & Provisioning	PIONEER [34]	Key based & software identity	Key based & Memory attestation	RTM* Measurements
Secure Communication	SAKE [42]	Key based	HMAC Correctness	IPC Proxy(between tasks)
Open Source	No	Yes	No	No

*EA-MPU: Execution Aware Memory Protection Unit, RTM: Root of Trust Measurements

circuit to drive the access rights of a software module. MAL ensures that the data in the protected data section is executable only if the program counter is in the text section of the executed module. The MAL circuit also ensures that other software modules can enter the text section through well-defined entry points.

- Remote Attestation: The symmetric key unique to a module and device is available to the software provider. The same key is generated on the device side and is accessible to the module only using processor instructions. The MAC generated using this key is sent across to the verifier in order to attest the software configuration.

- **Secure Communication:** When software provider requests for a data from the software module, the MAC over the data using symmetric key provides strong assurance that the data has been produced by the module on the specific device.

2.3 PUF-based Protocols

Aysu et al. proposed a mechanism to authenticate a device by cryptographically linking multiple PUFs on the PCB [1]. This prototype uses a μ Clinux operating system which contributes to almost 80% of software resources, which is quite expensive. This thesis provides an improvement over the approach by the use of SANCUS [38] on OpenMSP430 from the OpenCores project [43].

Several approaches use PUFs for software protection. Kohnhuser et al. uses SRAM PUFs to protect the software against modifications on a low-cost ARM-based microcontroller [44]. The PUF data is used to verify the authenticity of the device while hashing of the code measures the integrity of the software. Gora et al. proposed a system that utilized the PUF key to decrypt the actual software code before executing it [45]. Nithyanand et al. proposed a system based approach using IP-PUFs, where PUF is continuously associated with the control flow of the program [46]. The above approaches use PUFs as a trust anchor. The work described in this thesis relies on SANCUS [38] for the protection of the software, while the software particularly aims at the function of PUF enrollment.

Chapter 3

Design Optimizations: Processing component

In this chapter, we discuss the PUF Fusion Model [1] and how the prototype is optimized using a simpler processing component thereby providing an efficient solution for the PUF enrollment problem.

3.1 Existing Model

Aysu et al.[1] proposed a protocol of authenticating a remotely located PCB using its components and memory elements. These elements form the component-level PUFs characterized through the probability of a bit error e , and the probability d of a bit-flip between two instances of the same component. The paper discusses how multiple component PUFs can

be merged taking the error and inter-distance characteristics into consideration.

The paper [1] recommends the Reverse Fuzzy Extractor protocol [47] which implements mutual authentication between the server monitored by the infrastructure operator and the remotely installed device. The setup phase includes the secure enrollment of PUF data where the operator registers the device's public ID and the PUF data. The second phase is the authentication step where the server requests the device for the helper data and board ID. The server then reconstructs the noisy PUF using the enrolled PUF associated with the board ID. This step is succeeded by hash operations to verify the knowledge by both the parties.

3.1.1 System Architecture

Hardware components

The design comprises of a Nios-II softcore processor with several FPGA components on Altera DE2-115. The two PUFs implemented are SRAM PUF and Ring Oscillator PUF (RO PUF). The SRAM PUF includes a SRAM controller that measures the power-up state at a requested memory location and transfers it to the sender. The RO PUF includes multiple ring oscillators that exploit the frequency variations to produce a suitable PUF response.

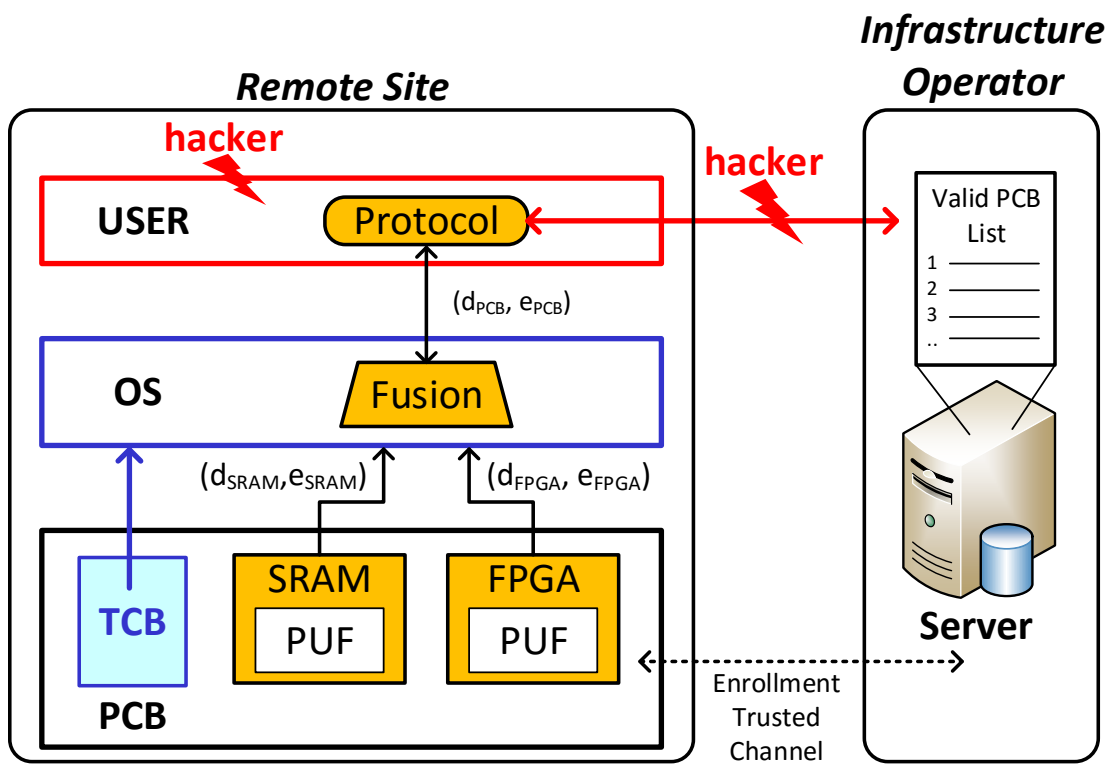


Figure 3.1: Device attestation using Fusion PUF [1]

Software components

The software component on the device included an μ CLinux operating system with kernel modules that had access to the PUF data. The kernel modules are assumed to be within a TCB and are accessible only to the application installed by the infrastructure provider. JTAG UART aids the exchange of requests and responses between the server and client applications. The protocol guarantees that the data received is indeed from the software installed on the particular device. Figure 3.2 gives an overview of the software architecture implemented in [1].

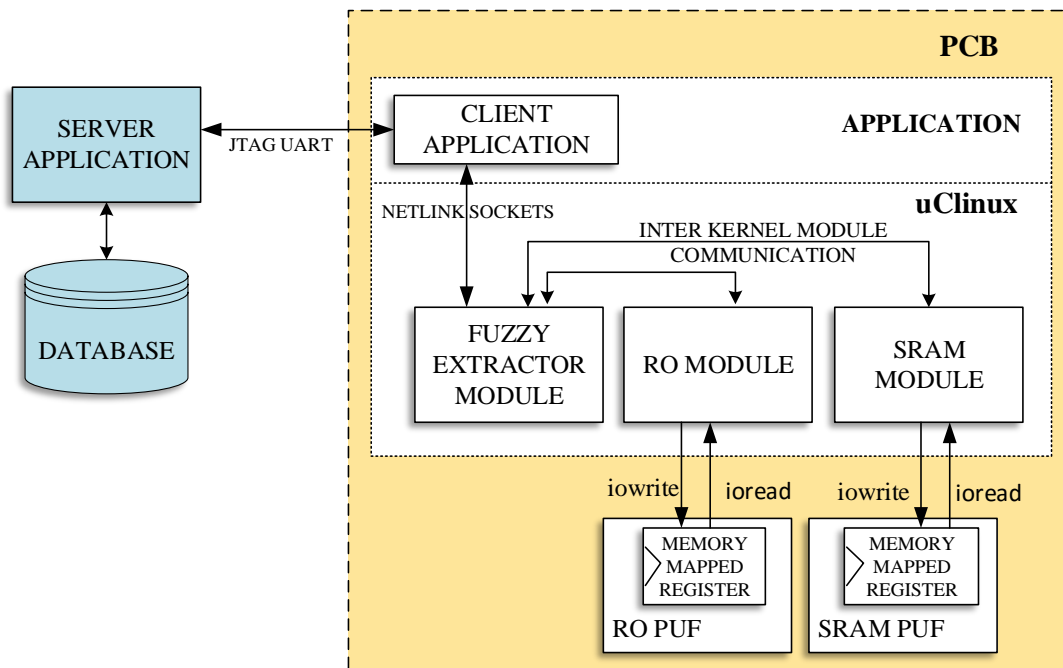


Figure 3.2: Block Diagram of the Software Architecture of Fusion PUF [1]

3.2 Optimization using microcontrollers

The μ CLinux kernel comprises of around 86.7% of the total of the software resources. This memory overhead can be eliminated by using a no OS system. A microcontroller with an efficient security architecture can form a suitable alternative to the system proposed by [1]. Section 2.2.2 provides an overview of the available solutions which can help to achieve a secure execution of PUF protocols within microcontrollers.

3.3 PUF Enrollment Solution

In this section, we discuss the construction of a solution for PUF enrollment.

3.3.1 Hardware architecture

System design

Figure 3.3 shows the hardware architecture of the prototype. The design includes the open-MSP430 of the OpenCores project [43] with the Memory Access Logic and SPONGENT-128/128/8 implemented by SANCUS [38]. The design also incorporates the SRAM PUF and Ring Oscillator PUFs from the PUF Fusion prototype [1].

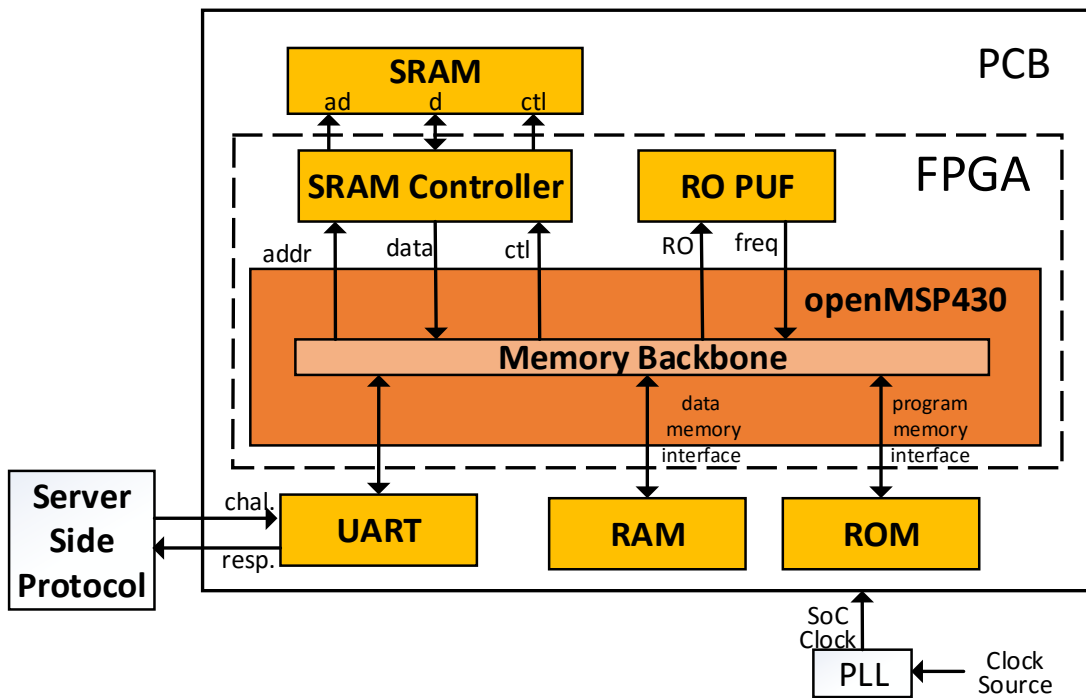


Figure 3.3: Block Diagram of the Hardware Architecture of PUF Enrollment Solution

PUF design and integration

The SRAM PUF design in the demonstrator uses an onboard static RAM (ISSI IS61WV102416BLL) and a FSM controller. The RO design comprises of 256 ROs to generate a 255-bit output. The openMSP430 architecture maps the SRAM PUF and the RO PUF within 512B of the peripheral memory space. The client software communicates with the hardware PUF modules using memory-mapped I/O.

3.3.2 Software architecture

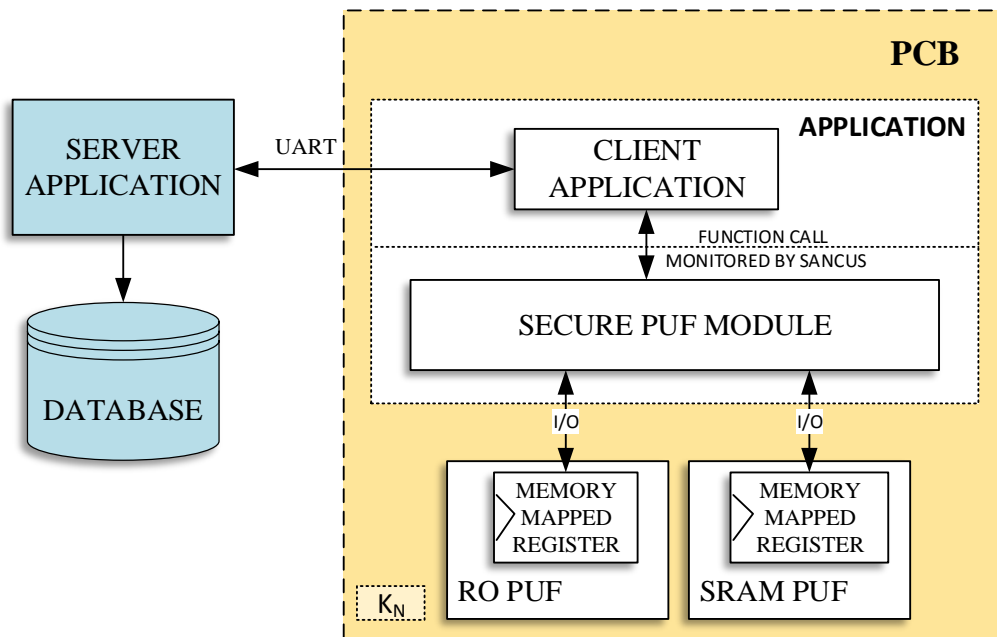


Figure 3.4: Block Diagram of the Software Architecture of PUF Enrollment Solution

Figure 3.4 shows the software architecture of the PUF enrollment prototype. It consists of

three main components. (i) the server application sends the challenge for PUF extraction, (ii) the secure PUF extraction module which can communicate with PUFs, and (iii) the client application that loads, protects and interacts with the secure module.

The application is compiled using modified LLVM provided by SANCUS. The PUF module benefits from the SANCUS by using certain special attributes to annotate global variables and functions.

3.3.3 Protocol

The infrastructure provider creates a client application that is loadable on all of its devices. It is possible because the vendor has the knowledge of all the elements that it needs to create the module key $K_{N,SP,SM}$ [38]. The server application initiates the extraction and sends a request to the client application. The host client application loads the secure module, enables protection and extracts the PUF data. It then evaluates the MAC over the data and returns it. The server application now compares the MAC with the locally computed MAC to identify if the returned PUF data is precisely from the software module installed by itself on its device.

3.3.4 Security Analysis

The prototype benefits from the security properties supported by SANCUS. The module needs to be thoroughly checked for accurate implementation since SANCUS doesn't provide

resistance against implementation breaks [38]. It is important to note that PUF extraction module relies on the isolation offered by the SANCUS which depends on a key stored in NVM. Our assumption is that the key is common or known for all the devices where the provider plans to install the PUF extraction software.

Chapter 4

Design Optimizations: Coding Algorithms

In this chapter, we discuss the SRAM Noise and entropy analysis and how this is used to create a suitable debiasing solution at low error rate expense.

4.1 SRAM Noise Pattern Analysis

4.1.1 Reason for noise in SRAM PUFs

A standard 6T SRAM consists of two nominally matched cross-coupled CMOS inverters and two access transistors. The CMOS manufacturing variations result in a mismatch between the inverter pairs causing the bias in the power up state. Along with the process changes, the

power state is also affected by the voltage supply, ambient temperature, aging effects. Fuzzy extractors are used as post-processing modules in PUF based techniques that can correct these noisy bits. But the helper data generated using fuzzy extractors can result in leakage of secret information in case of strongly biased PUF responses [48].

4.1.2 Biased and Noisy PUF Model

PUF Bias

Given a SRAM response of n bits, a bias of p indicates that any random bit in the n bit response is likely to be ‘0’ with a probability of p where $0 \leq p \leq 1$. An ideal unbiased PUF has $p = 0.5$.

PUF Noise

Delvaux et al. [17] considers error rate to be independent and identically distributed (i.i.d.) among PUF response bits and derives them using normally distributed manufacturing variability and noise components. In this thesis, we evaluate the error rate e based on the number of bit flips observed during multiple measurements for the same challenge at every bit position.

State flow diagram and Transition Matrix

Figure 4.1 shows an a generic PUF model created for an SRAM with error rate e and bias p . Each transition indicates the probability of switching to another symbol and how the bias p and the error rate e influences it. The symbol error rate is evaluated by adding all the outgoing transitions to another symbol. The symbol bias, on the other hand, is estimated by multiplying the bit dependent biases.

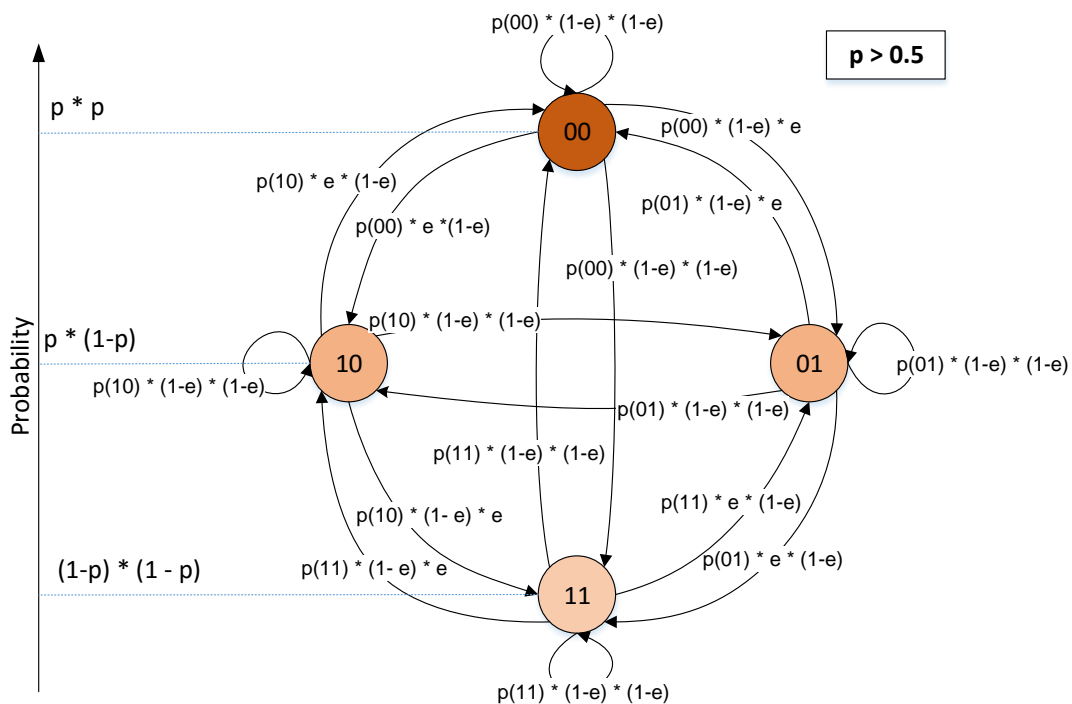


Figure 4.1: Error rate and probability when SRAM bits of noise n and bias p are grouped into 2-bit symbols

Table 4.1 provides an overview of the transition matrix that can be used to represent the

Table 4.1: Transition table for the SRAM data in fig. 4.1 where $p(00) = p * p$, $p(01) = p * (1 - p)$, $p(10) = (1 - p) * p$, $p(11) = (1 - p) * (1 - p)$.

	00	01	10	11
00	$p(00) * (1 - e) * (1 - e)$	$p(00) * (1 - e) * e$	$p(00) * e * (1 - e)$	$p(00) * e * e$
01	$p(01) * (1 - e) * e$	$p(01) * (1 - e) * (1 - e)$	$p(01) * e * e$	$p(01) * e * (1 - e)$
10	$p(10) * e * (1 - e)$	$p(10) * e * e$	$p(10) * (1 - e) * (1 - e)$	$p(10) * (1 - e) * e$
11	$p(11) * e * e$	$p(11) * e * (1 - e)$	$p(11) * (1 - e) * e$	$p(11) * (1 - e) * (1 - e)$

state flow diagram. The sum of all elements in a transition matrix evaluates to 1. Also, the sum of the off-diagonal elements equals the error rate e . The symbol probabilities are evaluated by adding the elements of the row corresponding to the symbol. For example, the sum of all elements of the row corresponding to symbol ‘00’ gives us $p(00)$.

4.1.3 Noise and probability distribution analysis using state flow diagrams

Figure 4.2 shows a state flow diagram created using simulated SRAM an error probability of 0.1, and a bias of 0.3 towards 0. The SRAM bits are read two bits at a time. Every outgoing transition indicates the likelihood of a 2-bit symbol switching to another 2-bit symbol upon successive accesses of the PUF using the same challenge. For example, the probability of switching from symbol ‘11’ to symbol ‘10’ is calculated as $p(\text{symbol}) * p(\text{error/no-error in the first bit}) * p(\text{error/no-error in the second bit})$ which results in 0.0784. Since, $p = 0.3$, probability of individual symbols are as follows : $p(00) = 0.09$, $p(01) = 0.21$, $p(10) = 0.21$, $p(11) = 0.49$. These values clearly indicate the bias towards the symbol ‘11’. The total

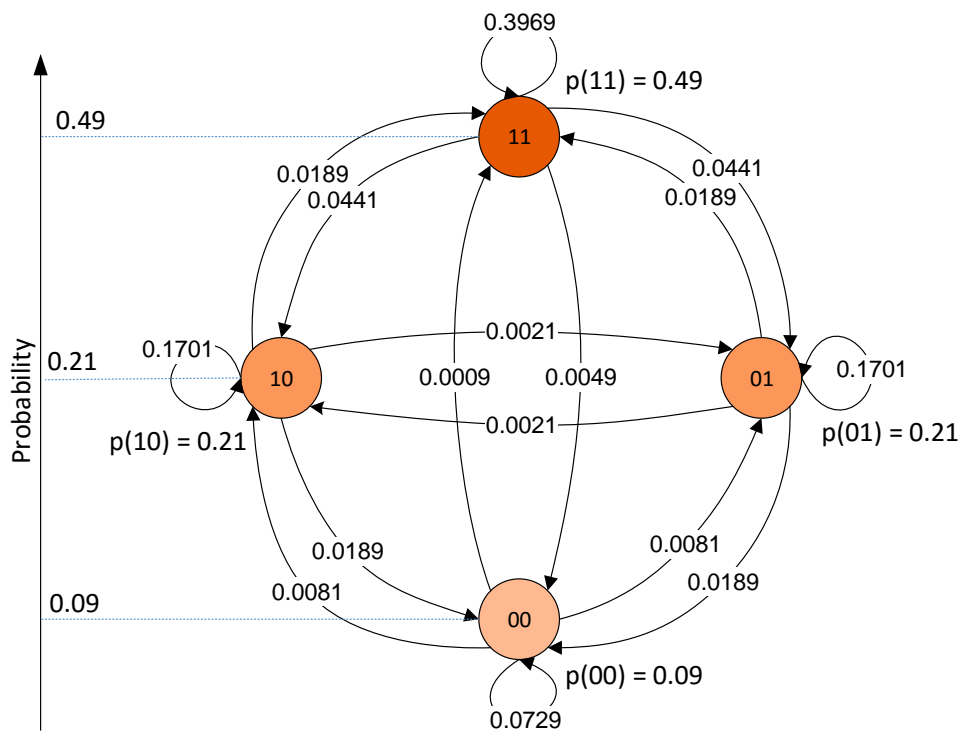


Figure 4.2: Error rate and probability when SRAM bits of $n=0.1$ and $p=0.3$ are grouped into 2-bit symbols

symbol error rate obtained by calculating the sum of all the outgoing transitions from one state to another state is 0.19.

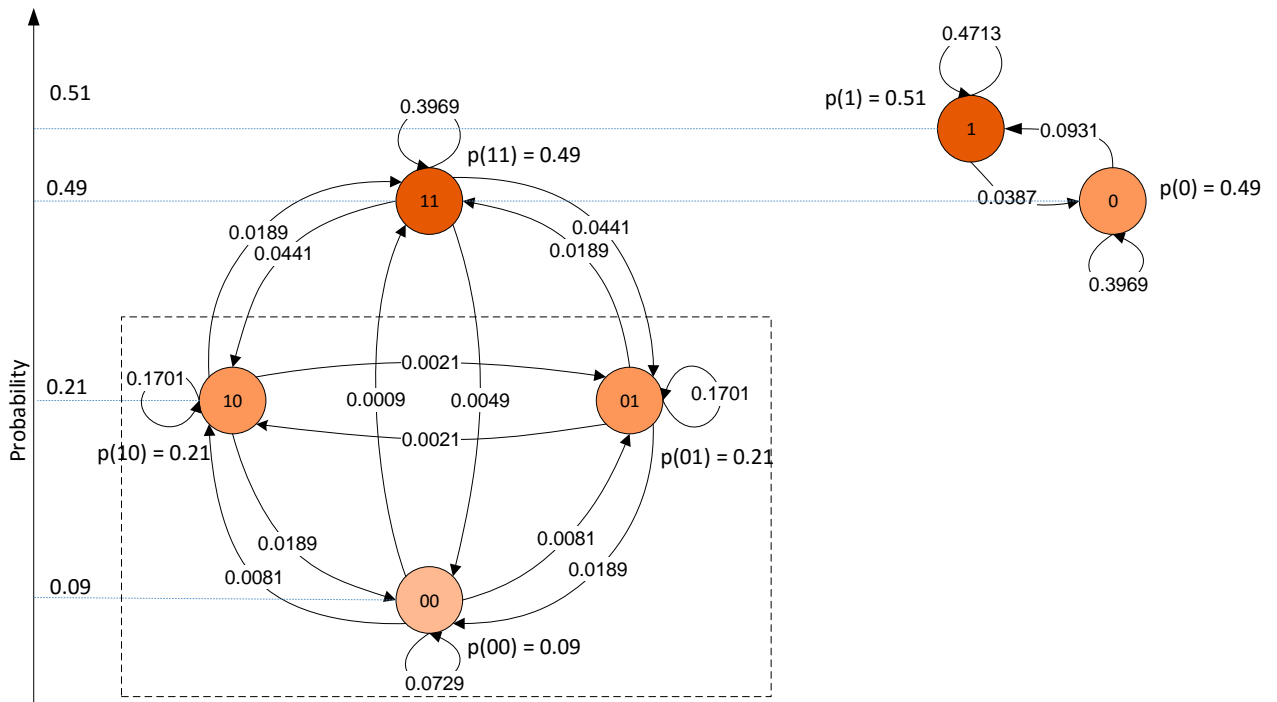


Figure 4.3: Grouping based on probability

A straight forward method of de-biasing the above model is by grouping symbols such that the probability distribution is uniform as shown in figure 4.3. But grouping the symbols solely based on probability doesn't ensure the best possible error rate which is 0.1318 in this case. Similarly, classifying the symbols entirely based on reducing the error rate (0.1) would not lead to an ideal distribution of symbols as seen in figure 4.4. Figure 4.5 shows another combination of symbols which is possible but doesn't yield best results with 0.18 bit error rate.

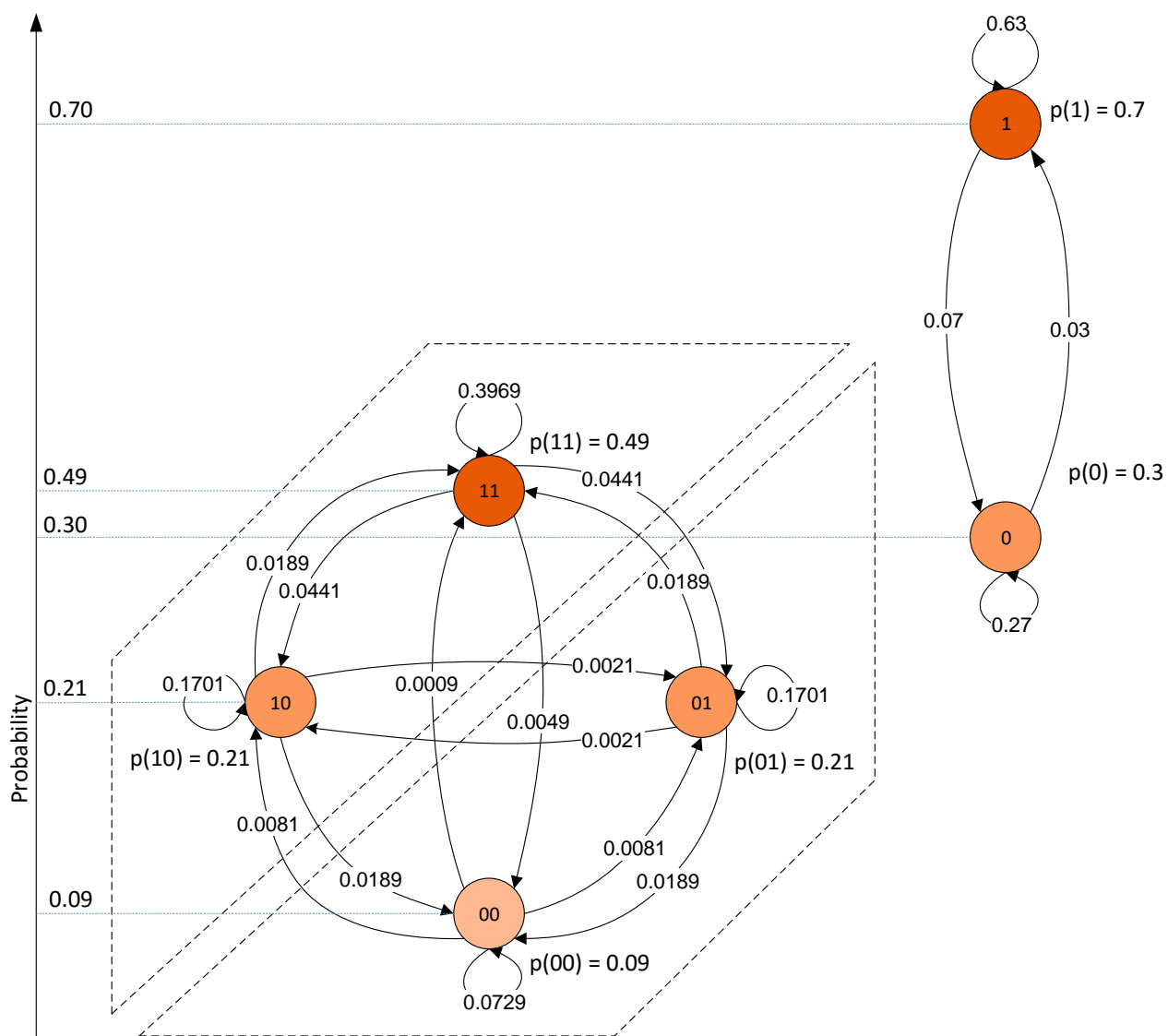


Figure 4.4: Grouping based on error rate

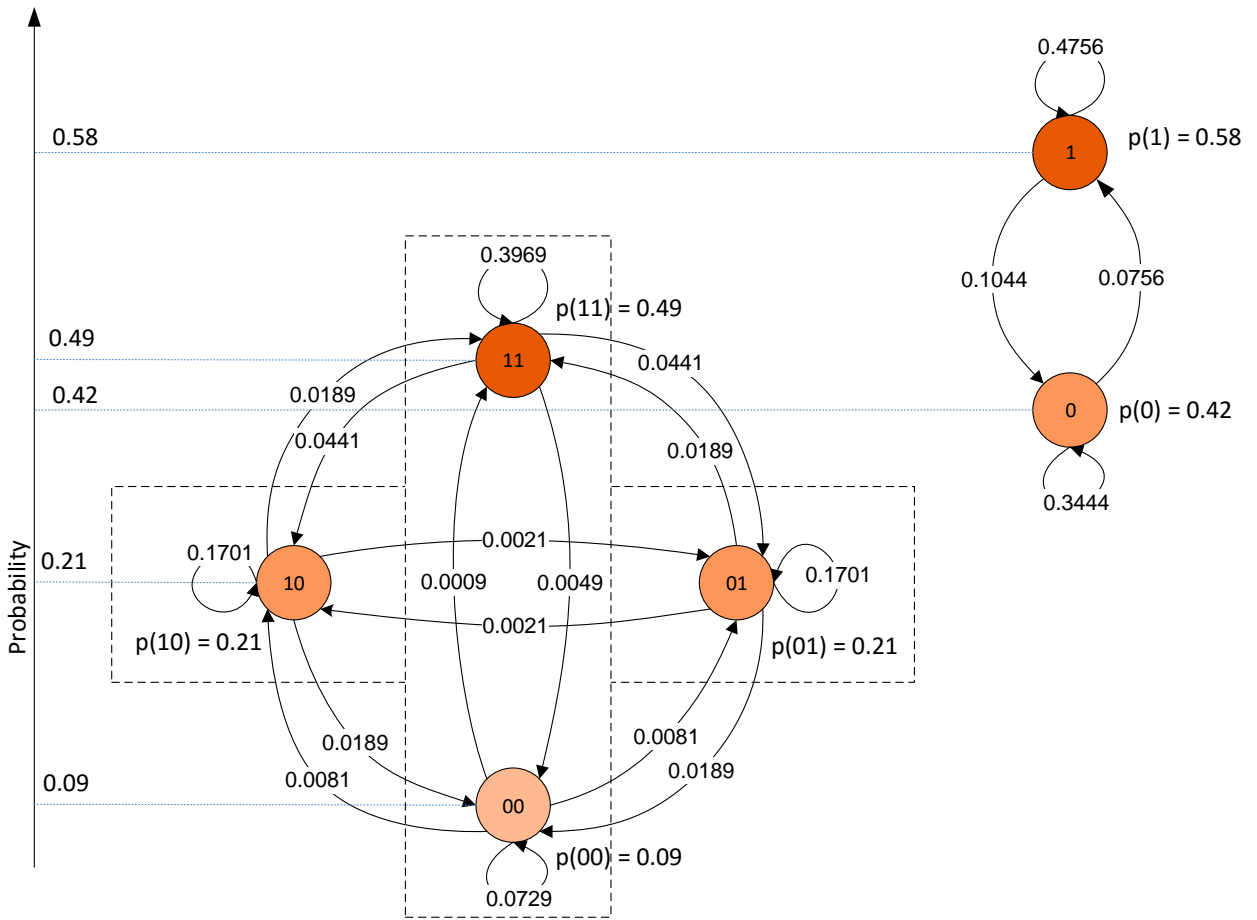


Figure 4.5: An alternate grouping option

This kind of grouping of symbols within a state flow model implies that it is possible to develop a suitable debiasing model by considering both the probability distributions and bit error rates. It is also important to note that the high probability symbols contribute the most to the error rate. For example, symbol error rate of switching from high probability symbol ‘11’ to other symbols is 0.0931 while symbol error rate of switching from low probability symbol ‘00’ to other symbols is 0.0171. As a result of which grouping symbols to reduce error rate will result in placing higher probability symbols in the same group causing an imbalance.

4.2 Symbol Encoding using Look Up Tables

This section explains how the state flow diagrams described above are utilized in the simultaneous analysis of error rate and probability distribution in achieving a de-biasing solution.

4.2.1 Construction of Transition Tables

The state flow diagrams provide an insight to how bits are related and can be combined to obtain a rather uniform distribution while optimizing the bit error rate. It is important to note that the error transitions are not exactly symmetric in the case of actual SRAM measurements. Figure 4.6 shows that symbol ‘10’ has a higher probability compared to other 2-bit symbols for a SRAM on a SASEBO board.

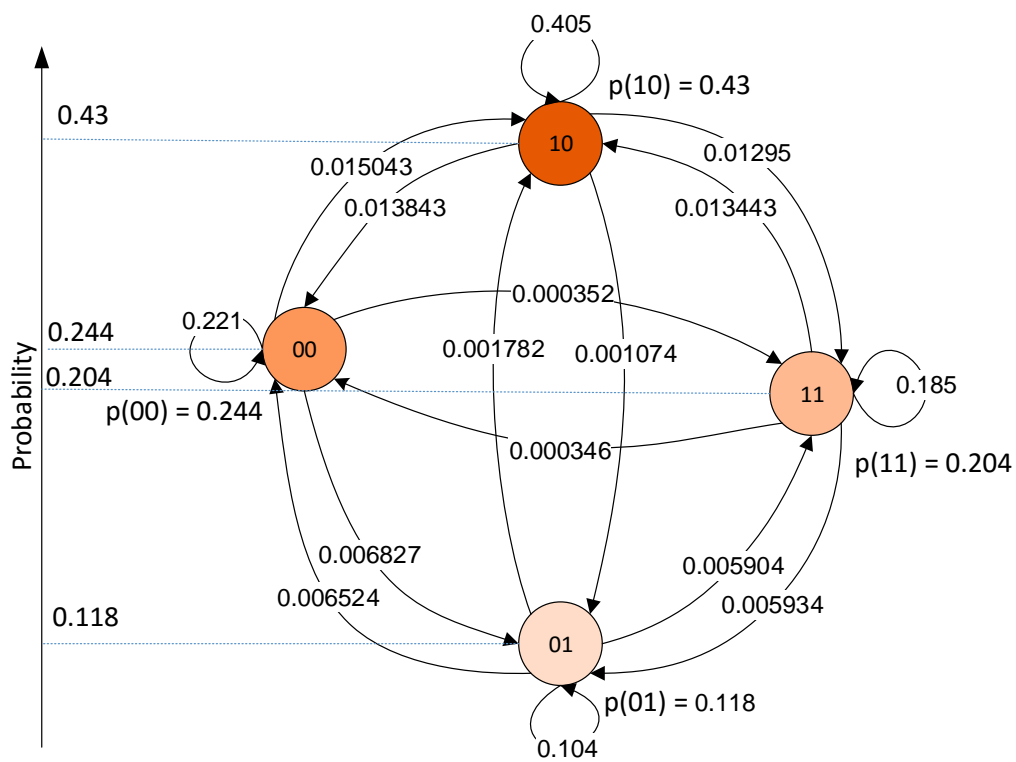


Figure 4.6: State flow diagram for SRAM bits from a SASEBO board

The state flow diagrams observed can also be analyzed using a transition table shown in table 4.2. From the figure, the probability of switching from symbol ‘00’ to symbol ‘10’ is 0.01504. The total error rate is calculated by adding all the off-diagonal elements while the probability of a symbol is measured by adding the row elements for that symbol.

Table 4.2: Transition table for the SRAM data in fig. 4.6

	00	01	10	11
00	0.221754	0.006827	0.01504	0.000352
01	0.006524	0.104181	0.001782	0.005904
10	0.013843	0.001074	0.405448	0.01295
11	0.000346	0.005934	0.013443	0.184595

4.2.2 Typical grouping approaches

Priority: Uniform probability distribution

Achieving a uniform probability distribution among the group of symbols is a partition problem which can be solved using dynamic programming or other approaches. The resulting distribution ensures a high min-entropy among the resulting bits but at the expense of bit errors. Algorithm 1 provides a detailed sequence of operations involved in clustering based on probabilities of individual symbols.

Algorithm 1: Clustering algorithm to achieve an uniform probability distribution

Function *SymbolClusteringBasedOnProbability*(n, k)

Input : Input number of bits n , and output number of bits k

Output: Labels for each of the n -bit symbols

Generate the input n -bit transition matrix for the SRAM type;

Create n initial cluster nodes with each symbol in it's own cluster;

$num_clusters \leftarrow n$;

while $num_clusters < k$ **do**

for $i \leftarrow 1$ **to** $num_clusters$ **do**

 | $probability[i] \leftarrow \text{Probability}(cluster[i])$;

end

$sorted_indices \leftarrow \text{SortDescending}(probability)$;

$new_cluster_index \leftarrow \text{Min}(sorted_indices(1), \dots, sorted_indices(k))$;

$\text{MergeClusters}(cluster(new_cluster_index), cluster(k+1))$;

$num_clusters \leftarrow num_clusters - 1$;

end

Priority: Reduction of error rate

A low error rate can be achieved by grouping symbols which are most likely to confuse with each other. Since the symbols with higher probability have a greater tendency to confuse among each other, the resulting distribution will result in a low min-entropy. Algorithm 2 provides a detailed sequence of operations involved in clustering based on error rate associated with every symbol.

Algorithm 2: Clustering algorithm to achieve a low symbol error rate

Function *SymbolClusteringBasedOnError*(n, k)

Input : Input number of bits n , and output number of bits k
Output: Labels for each of the n -bit symbols

Generate the input n -bit transition matrix for the SRAM type;
 Create n initial cluster nodes with each symbol in it's own cluster;

$num_clusters \leftarrow n$;
while $num_clusters < k$ **do**
 | **for** $i \leftarrow 1$ **to** $num_clusters - 1$ **do**
 | | **for** $j \leftarrow i$ **to** $num_clusters$ **do**
 | | | $distance[i][j] \leftarrow \text{ClusterDistance}(cluster(i), cluster(j))$;
 | | **end**
 | **end**
 | $[index_i, index_j] = \text{MinDistance}(distance)$;
 | $\text{MergeClusters}(cluster(index_i), cluster(index_j))$;
 | $num_clusters \leftarrow num_clusters - 1$;
end

4.2.3 Grouping with a balance between probability distribution and error rate

In this thesis, we propose an alternate algorithm that jointly considers the effect of symbol distribution on min-entropy and bit error rate. Algorithm 3 provides the detailed sequence of operations involved. At every step, the error rate and probability distributions are pre-computed before combining the two symbols. This comparison is made to ensure that balance in probability is achieved while maintaining a low effective error rate. The labels are then used to create a look-up table to generate the de-biased PUF-responses. The advantage of this approach is that the clustering combinations are pre-calculated before deploying it on the devices. Hence, more advanced machine-learning based approaches can be used to analyze

and efficiently create cluster combinations.

Algorithm 3: Clustering algorithm that jointly optimizes error rate and min-entropy

Function *SymbolClusteringWithProbabilityErrorBalance*(n, k)

Input : Input number of bits n , and output number of bits k

Output: Labels for each of the n -bit symbols

Generate the input n -bit transition matrix for the SRAM type;

Create n initial cluster nodes with each symbol in it's own cluster;

$num_clusters \leftarrow n$;

while $num_clusters < k$ **do**

for $i \leftarrow 1$ **to** $num_clusters$ **do**

$probability[i] \leftarrow \text{Probability}(cluster[i])$;

end

$sorted_indices \leftarrow \text{SortDescending}(probability)$;

for $i \leftarrow 1$ **to** k **do**

$new_cluster \leftarrow \{cluster(sorted_indices(1)), cluster(sorted_indices(k + 1))\}$;

$noise_k[i] \leftarrow \text{CalculateNoise}(new_cluster)$;

$prob_dist_k[i] \leftarrow \text{CalculateProbabilityDistribution}(new_cluster)$;

end

$sorted_position \leftarrow \text{SortAscending}(noise_k)$;

for $i \leftarrow 1$ **to** k **do**

$j = sorted_position(i)$;

if $\text{Max}(prob_dist_k[j]) \leq 1/k$ **then**

$index \leftarrow j$;

break;

end

end

$\text{MergeClusters}(cluster(sorted_indices(index)), cluster(k+1))$;

$num_clusters \leftarrow num_clusters - 1$;

end

4.2.4 Change in error correcting parameters

The algorithm suggested in this thesis considers the tradeoff between min-entropy and error rate. It efficiently provides an increase in min-entropy at the cost of a greater number of bits read from the SRAM PUF. But considering the large size of SRAM present on any off the shelf component and requirement of no additional hardware makes it feasible for its implementation. Also, the approach makes it possible for certain SRAMs to create PUF based security solutions which inherently did not possess good min-entropy or error rate as observed in [48].

4.2.5 Security Analysis

The look-up table is unique for a particular device and is known only to the infrastructure provider. As a result, the look-up tables add an extra layer of randomness in a PUF-authentication systems where the adversary is not aware of the expected PUF response in spite of having physical contact with the device.

The look-up tables can be used along with SANCUS [38] to provide an improved enrollment solution. SANCUS protects the data section containing the look-up table making impossible for the adversary to access the code region or guess its implementation.

Chapter 5

Experimental Results and Discussion

In this chapter, we discuss the implementation costs and performance of SANCUS based enrollment solution as well as the effect of look-up tables.

5.1 Enrollment Solution

5.1.1 Implementation Cost and Performance Analysis

The enrollment solution is demonstrated using an Altera DE2-115 FPGA board. The hardware components of the device were implemented using the Altera Quartus II toolchain version 15.0. Table 5.1 provides a comparison of hardware utilization of FPGA components among the three approaches. The table shows that the SANCUS based solution is more expensive compared to the simple OpenMSP430 based solution due to the comparatively

high hardware cost provided by the hardware-only TCB implementation of SANCUS. It is also important to note that the presence of a TCB was assumed in the case of the PUFFusion with NIOS processor-based solution. Hence additional hardware or software based TCB solution is required.

Table 5.2 provides an overview of memory footprint in case of the three approaches. A bare-metal environment proves to offer a lower memory utilization when compared to μ Clinux based solution. SANCUS adds a few additional bytes of overhead due to the modifications required for registering a new protected module.

Table 5.1: Hardware utilization in terms of logic cells

Components	NIOS + PUFFusion	SANCUS + MSP430 + PUFFusion	MSP430 + PUFFusion
Processor	2254	4705	2108
MM interconnect	671	0	0
RO-PUF	4385	4407	4407
SRAM controller	5	35	36
SDRAM controller	327	0	0
JTAG-UART	146	161	163
PLL	8	0	0
Total	7796	9308	6714

The protocol takes about 455.148 microseconds for the complete operation. The device returns the PUF data and the MAC associated with the data in 307.506 microseconds.

Table 5.2: Software utilization in bytes

Components	NIOS + PUFFusion	SANCUS + MSP430 + PUFFusion	MSP430 + PUFFusion
RO-PUF	38820	484	484
SRAM PUF	35128	136	136
SANCUS Overhead	0	112	0
Board Application	35360	4060	4060
uClinux Kernel	1475069	0	0
Total	1,584,377	4792	4680

5.2 Look up tables

5.2.1 Implementation

We analyzed the SRAM measurements of DE2, SASEBO, and PIC32, as shown in table 5.3. For each of the board type, an n-bit transition matrix was generated examining n bits at a time. This transition matrix was later subjected to the clustering algorithm to evaluate the appropriate mappings for each of the symbols. The SRAM bits are read again, mapped to appropriate symbols, and the min-entropy and bit error rate are evaluated for the mapped SRAM response.

5.2.2 Performance Evaluation

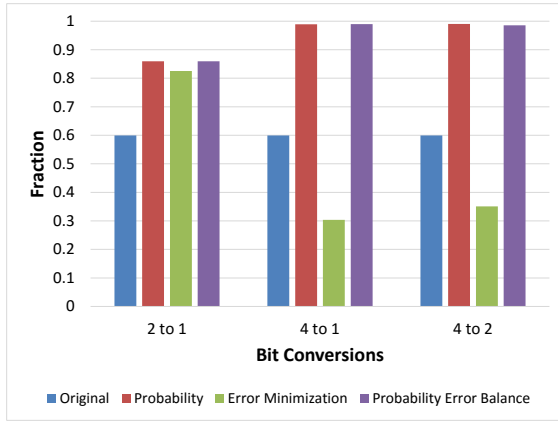
Figures 5.1 - 5.3 give a comparison of min-entropy and error rate values for simulated SRAM values. It is evident that the min-entropy improves in the case of probability based approach

Table 5.3: SRAM Datasets

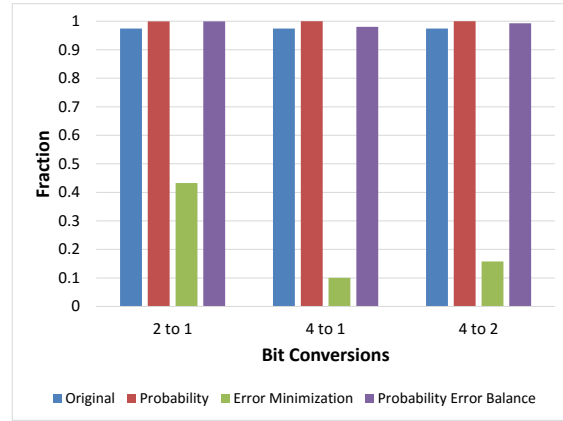
Board name	Number of boards	Number of measurements
DE2	22	5
SASEBO	90	11
PIC32	33	9

and drastically decreases in the case of error minimization based approach. The clustering approach proposed by this thesis tries to achieve a balance between the probability and the error rate providing a feasible de-biasing solution.

Figures 5.4 - 5.12 present the comparison of min-entropy and error rate values for SASEBO, DE2 and PIC32 boards for various bit conversions. It is evident that the min-entropy in the case of clustering method based on probability error balance is greater than the initial entropy observed in the SRAM data. Also, the symbol error rate in the case of the clustering method proposed by the thesis is lesser than the original symbol rate. Also, the min-entropy and error rate of the mapped SRAM measurements closely resemble the results from the transition matrix. This indicates that applying a look table on the SRAM data at runtime results in a PUF response expected by the algorithm, i.e. there is a decrease in symbol error rate and an increase in min-entropy.

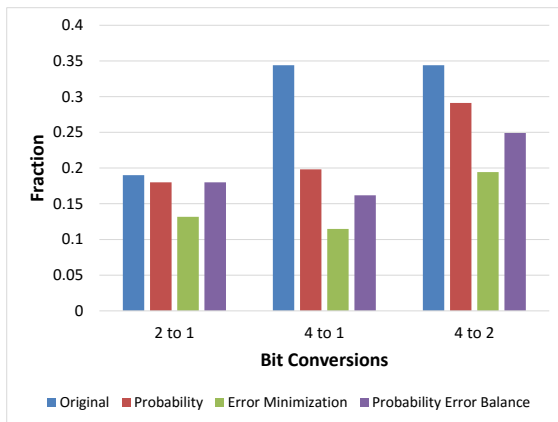


(a) Evaluation of the transition matrix for $p = 0.3$ and $n = 0.1$

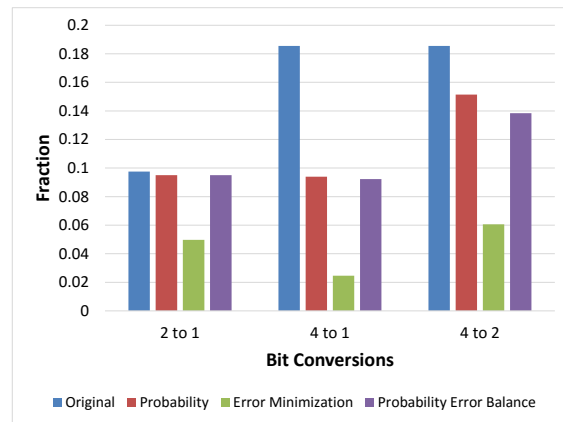


(b) Evaluation of the transition matrix for $p = 0.49$ and $n = 0.05$

Figure 5.1: Comparison of min-entropy values for SRAM for simulated p and n values

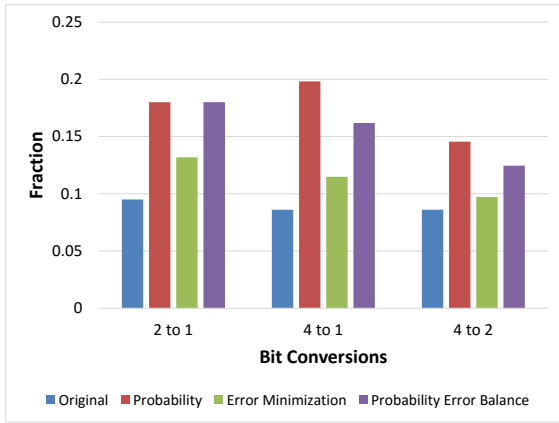


(a) Evaluation of the transition matrix for $p = 0.3$ and $n = 0.1$

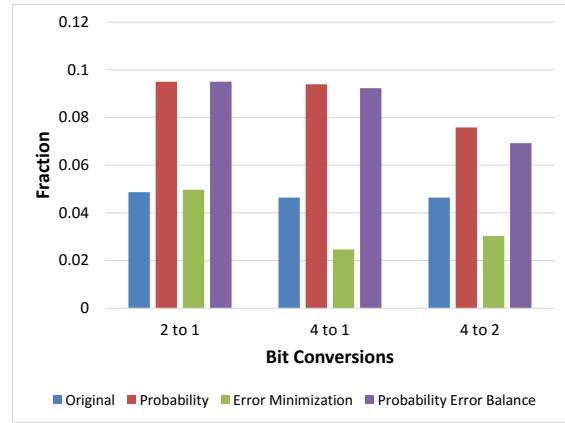


(b) Evaluation of the transition matrix for $p = 0.49$ and $n = 0.05$

Figure 5.2: Comparison of symbol error rate values for SRAM for simulated p and n values

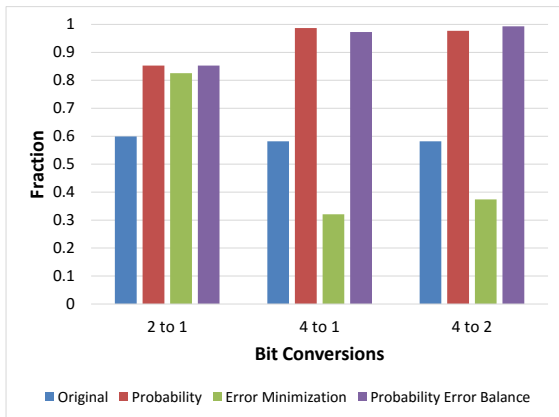


(a) Evaluation of the transition matrix for $p = 0.3$ and $n = 0.1$

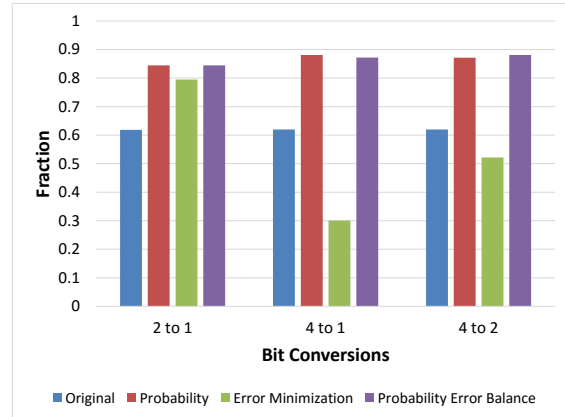


(b) Evaluation of the transition matrix for $p = 0.49$ and $n = 0.05$

Figure 5.3: Comparison of bit error rate values for SRAM for simulated p and n values

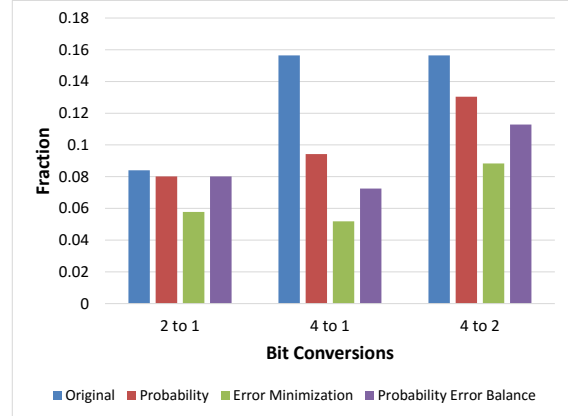
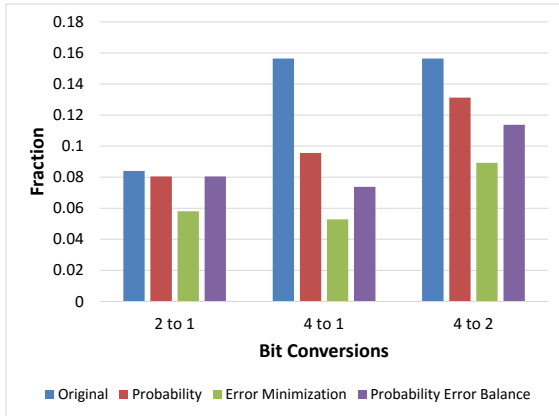


(a) Evaluation of the transition matrix



(b) Measurements using the mapped SRAM bits

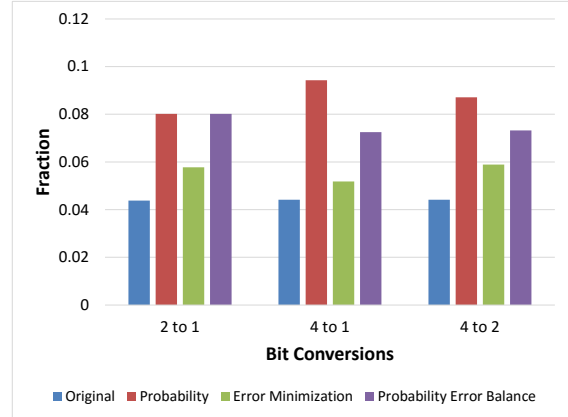
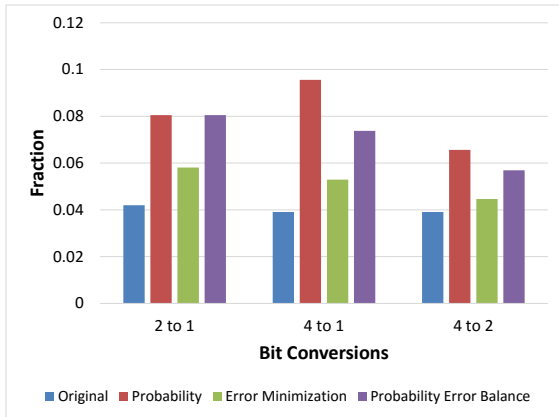
Figure 5.4: Comparison of min-entropy values for SRAM on SASEBO board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

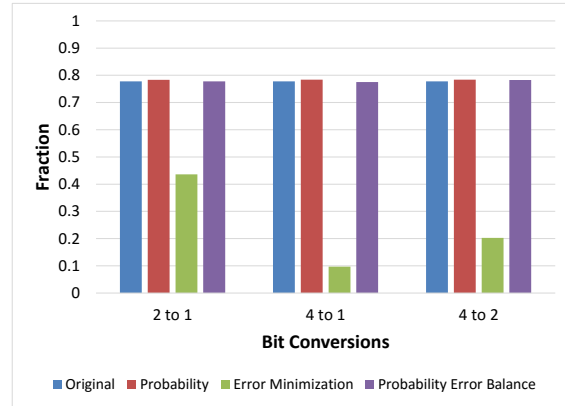
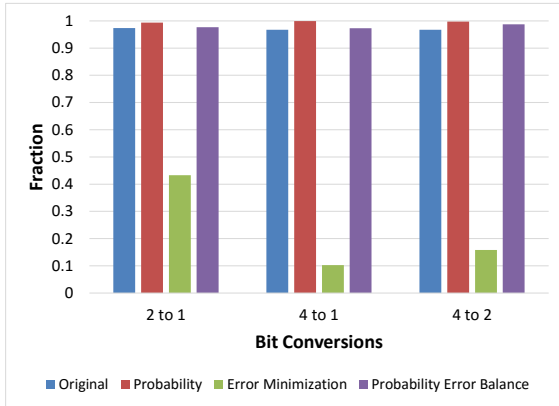
Figure 5.5: Comparison of symbol error rate values for SRAM on SASEBO board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

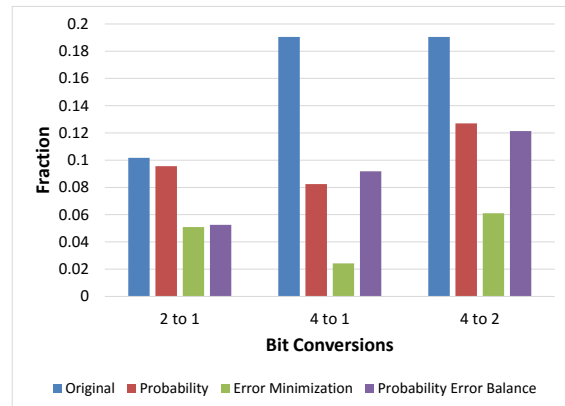
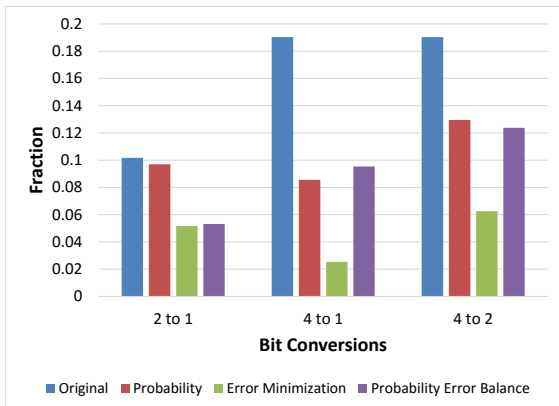
Figure 5.6: Comparison of bit error rate values for SRAM on SASEBO board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

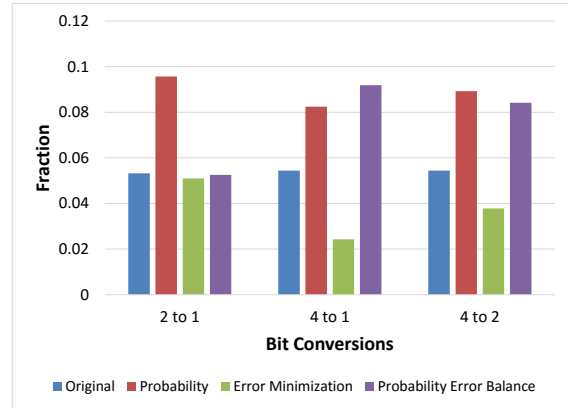
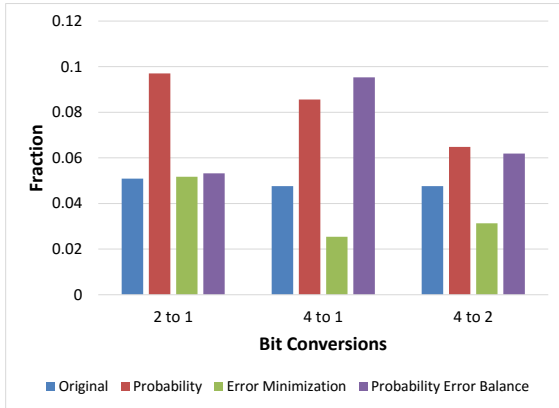
Figure 5.7: Comparison of min-entropy values for SRAM on DE2 board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

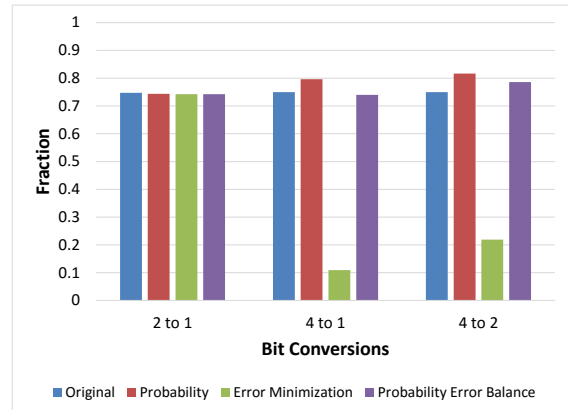
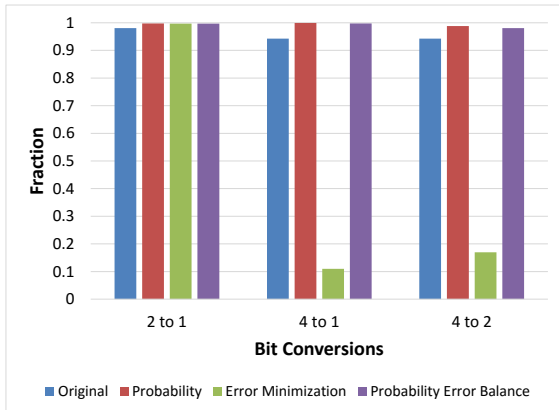
Figure 5.8: Comparison of symbol error rate values for SRAM on DE2 board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

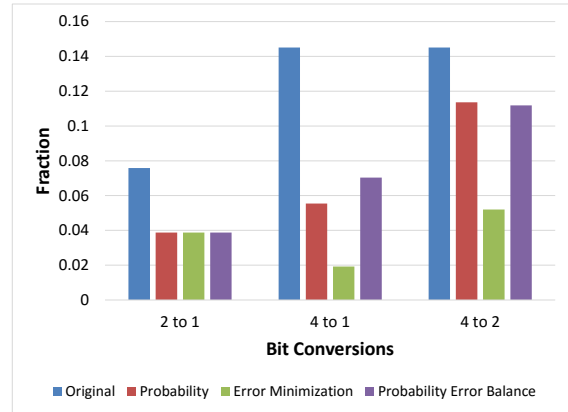
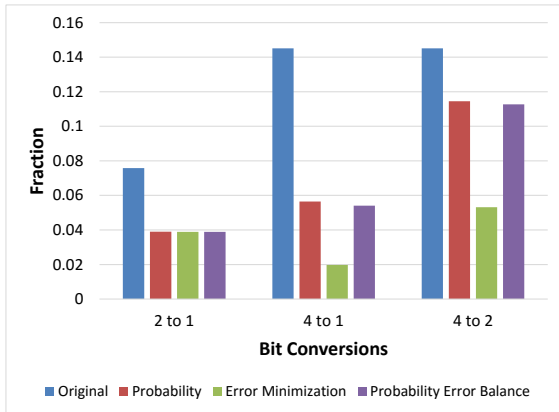
Figure 5.9: Comparison of bit error rate values for SRAM on DE2 board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

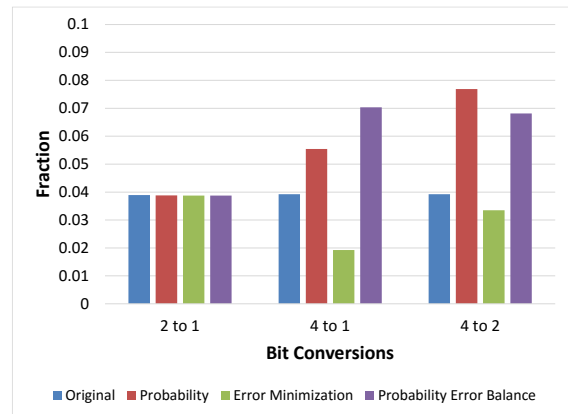
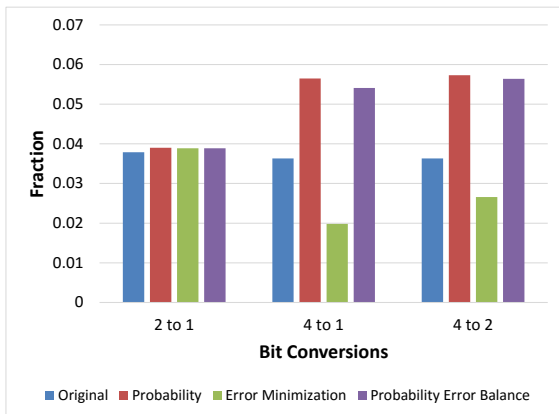
Figure 5.10: Comparison of min-entropy values for SRAM on PIC32 board



(a) Evaluation of the transition matrix

(b) Measurements using the mapped SRAM bits

Figure 5.11: Comparison of symbol error rate values for SRAM on PIC32 board



(a) Evaluation of the transition matrix

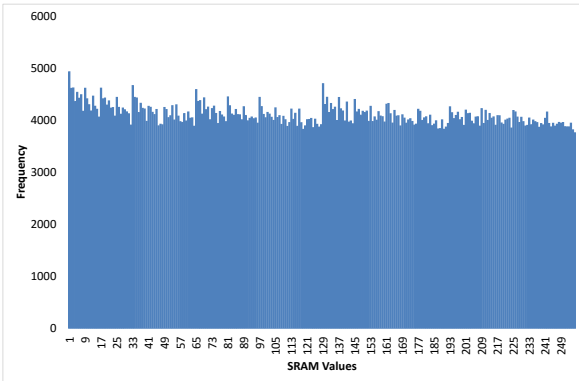
(b) Measurements using the mapped SRAM bits

Figure 5.12: Comparison of bit error rate values for SRAM on PIC32 board

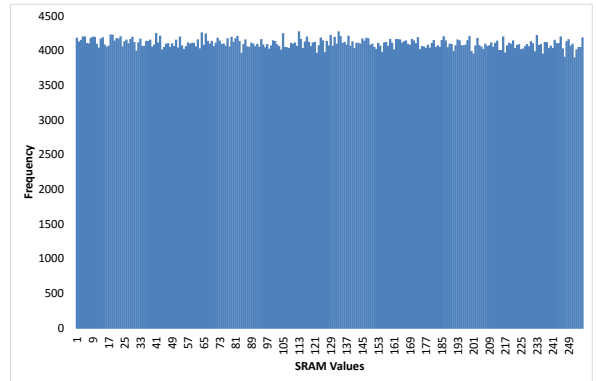
Table 5.4 shows that using a clustering method with a balance of error rate and probability results in the reduction of BCH block size but an increase in the number of blocks. Many of the PUF based solutions use fuzzy extractors with complex decoding logic on the device side [49, 50]. In such cases, a reduction of BCH block size can be advantageous. But the clustering algorithms do not work well for SRAM data from DE2 boards and PIC32 boards. A possible reason for this behavior is that the bias observed in SRAM data from SASEBO is greater than that of SRAM values from DE2 and PIC32 boards. This bias is evident from the histograms of DE2 and SASEBO boards in the figures 5.13 and 5.14. It implies that the clustering approaches work well only in the case of SRAM input bits with a significant bias.

Table 5.4: Comparison of the three approaches in terms of BCH parameters for an expected entropy of 128 bits and FRR of 10^{-3}

Board type	Bit Conversion	Priority	PUF size	BCH parameters(n,k,t)	BCH blocks
SASEBO	1 to 1	Original	4599	(511,211,41)	9
SASEBO	2 to 1	Probability	9180	(255, 47, 42)	36
		Error rate	11176	(127, 29, 21)	88
		Probability Error balance	9180	(255, 47, 42)	36
SASEBO	4 to 1	Probability	9180	(255, 45, 43)	36
		Probability Error balance	11684	(127, 22, 23)	92
SASEBO	4 to 2	Probability	10200	(255, 47, 42)	40
		Probability Error balance	9652	(127, 22, 23)	76
DE2	1 to 1	Original	21336	(127, 29, 21)	168
DE2	2 to 1	Probability Error Balance	42672	(127, 29, 21)	336
PIC32	1 to 1	Original	1020	(255, 107, 22)	4
PIC32	2 to 1	Probability	2040	(255, 107, 22)	4
		Error rate	2040	(255, 107, 22)	4
		Probability Error balance	2040	(255, 107, 22)	4
PIC32	4 to 1	Probability	21336	(127, 29, 21)	42
PIC32	4 to 2	Probability	420240	(255, 47, 42)	412
		Probability Error balance	22484	(511, 121, 58)	11

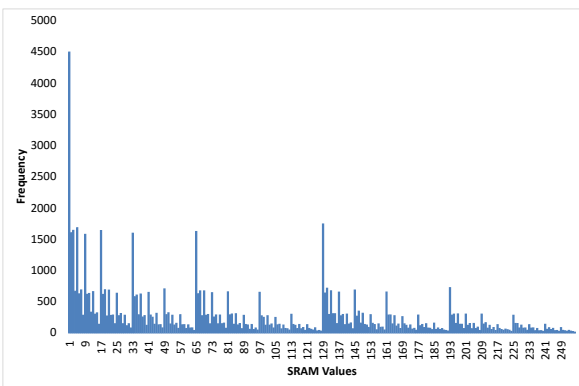


(a) Before clustering

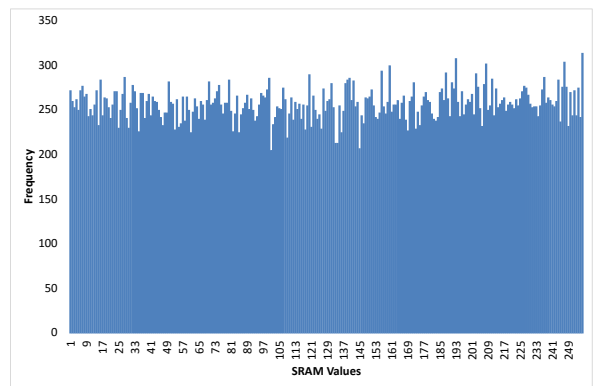


(b) After clustering

Figure 5.13: Histogram of SRAM byte values of DE2 board upon 2 bit to 1 bit clustering



(a) Before clustering



(b) After clustering

Figure 5.14: Histogram of SRAM byte values of SASEBO board upon 4 bit to 2 bit clustering

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis proposed two resource optimization techniques for PUF based device authentication protocols. The first approach is to optimize the processing element and add a protected execution environment to create a secure enrollment solution for PUFs. The secure architecture SANCUS guarantees the security of the PUF extraction software by making it impenetrable because of its secure loading and Memory Access Logic based implementation.

The helper data generated from a biased PUF response can result in leakage of entropy. The second part of this thesis offers a suitable debiasing solution which jointly optimizes the min-entropy and error rate. This form of combined optimization is necessary because favoring either min-entropy or the error rate can result in the deviation from the expected PUF

behavior. The implementation proposed by this thesis comes at the expense of PUF size. But considering the massive size of SRAM available in commercial off-the-shelf components, this method offers feasible solutions for strongly biased PUFs where a low min-entropy would make PUF based solutions unrealizable in such devices.

6.2 Future Work

This thesis concentrates on optimizing the min-entropy and error rate while converting the symbols from two bits to one bit, four bits to one bit, and four bits to two bits. This approach can be extended to analyze the behavior in case of a higher number of bits. Also, a careful decision must be taken during symbol reassignment when the target number of symbols are greater than one. For example, two-bit symbols which tend to often confuse with each other need to have a Hamming distance of one instead of two. This Hamming distance can affect the overall bit error rate.

PUF based authentication methods usually employ BCH for noise correction. Since our clustering approach deals with PUF data symbols, it is better to consider strategies that deal with coding algorithms that work better with symbols. There have been several other schemes that employ symbol based decoding with PUF data [51, 23]. It is interesting to analyze how these approaches can improve the overall performance of target PUF responses generated using look-up tables.

The clustering approach suggested in this thesis considers a simple method of joint opti-

mization of min-entropy and error-rate. Machine learning based approaches can be used to arrive at the sweet spot between min-entropy and error-rate. These computationally intensive methods are feasible because the analysis is not run-time and happens well before the implementation is deployed on the resource-constrained device.

Bibliography

- [1] A. Aysu, S. Gaddam, H. Mandadi, C. Pinto, L. Wegryn, and P. Schaumont, “A design method for remote integrity checking of complex PCBs,” pp. 1517–1522.
- [2] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical One-Way Functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.
- [3] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-Up SRAM state as an identifying fingerprint and source of true random numbers,” *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.
- [4] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, “FPGA Intrinsic PUFs and Their Use for IP Protection,” *Lecture Notes in Computer Science*, vol. 4727, pp. 63–80, 2007.
- [5] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, “End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol,” *Lecture Notes in Computer*

- Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9293, pp. 556–576, 2015.
- [6] R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. V. der Sluis, and V. van der Leest, “Experimental evaluation of Physically Unclonable Functions in 65 nm CMOS,” *European Solid-State Circuits Conference*, pp. 486–489, 2012.
- [7] P. Simons, E. Van Der Sluis, and V. Van Der Leest, “Buskeeper PUFs, a promising alternative to D Flip-Flop PUFs,” *Proceedings of the 2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012*, pp. 7–12, 2012.
- [8] Y. Su, J. Holleman, and B. Otis, “A1.6pJ/blt 96% stable chip-ID generating circuit using process variations,” *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, pp. 15–17, 2007.
- [9] R. Maes, P. Tuyls, and I. Verbauwhede, “Intrinsic PUFs From Flip-Flops on Reconfigurable Devices,” *Workshop on Information and System Security*, no. 71369, pp. 1–17, 2008.
- [10] S. Katzenbeisser, Ü. Kocaba, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “PUFs: Myth, fact or busted? A security evaluation of Physically Unclonable Functions (PUFs) cast in silicon,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7428 LNCS, pp. 283–301, 2012.

- [11] J. Lee, D. L. D. Lim, B. Gassend, G. Suh, M. V. Dijk, and S. Devadas, “A technique to build a secret key in integrated circuits for identification and authentication applications,” *Proceedings of 2004 Symposium on VLSI Circuits.*, pp. 176–179, 2004.
- [12] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” *Proceedings - Design Automation Conference*, pp. 9–14, 2007.
- [13] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, “Modeling attacks on physical unclonable functions,” *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, p. 237, 2010.
- [14] J. L. Zhang, G. Qu, Y. Q. Lv, and Q. Zhou, “A survey on silicon PUFs and recent advances in ring oscillator PUFs,” *Journal of Computer Science and Technology*, vol. 29, no. 4, pp. 664–678, 2014.
- [15] R. Maes and V. Van Der Leest, “Countering the effects of silicon aging on SRAM PUFs,” *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014*, pp. 148–153, 2014.
- [16] R. Maes and I. Verbauwhede, *Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions*. 2010.
- [17] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, “Helper data algorithms for puf-based key generation: Overview and analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 889–902, 2015.

- [18] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data,” vol. 38, no. 1, pp. 97–139, 2008.
- [19] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A. R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7397 LNCS, pp. 374–389, 2012.
- [20] M. Bhargava, C. Cakir, and K. Mai, “Reliability enhancement of bi-stable PUFs in 65nm bulk CMOS,” *Proceedings of the 2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012*, pp. 25–30, 2012.
- [21] M. Cortez, S. Hamdioui, V. Van Der Leest, R. Maes, and G. J. Schrijen, “Adapting voltage ramp-up time for temperature noise reduction on memory-based PUFs,” *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*, pp. 35–40, 2013.
- [22] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. M. Tehranipoor, “Bit selection algorithm suitable for high-volume production of SRAM-PUF,” *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014*, pp. 101–106, 2014.
- [23] M. D. Yu, M. Hiller, and S. Devadas, “Maximum-Likelihood Decoding of Device-Specific Multi-Bit Symbols for Reliable Key Generation,” *Hardware Oriented Security and Trust*

(HOST), 2015 IEEE International Symposium on. IEEE, 2016.

- [24] M. D. Yu and S. Devadas, “Secure and Robust Error Corrections for PUF,” *Institute of Electrical and Electronics Engineers*, 2010.
- [25] R. Maes, V. Van Der Leest, E. Van Der Sluis, and F. Willems, “Secure key generation from biased PUFs,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9293, pp. 517–534, 2015.
- [26] J. Ekberg, “Mobile Trusted Computing,” *Fruct.Org*, 2011.
- [27] S. W. Smith, “Outbound Authentication for Programmable Secure Coprocessors,” *Esorics*, vol. 2502, pp. 72–89, 2002.
- [28] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. a. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, “Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems,” *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems - ASPLOS XIII*, p. 2, 2008.
- [29] “Trusted Computing Group: Tpm main specification,” <http://www.trustedcomputing-group/>.

- [30] J. S. Dvoskin and R. B. Lee, “Hardware-rooted trust for secure key management and transient trust,” *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 389–400, 2007.
- [31] B. Pfizmann, J. Riordan, C. Stueble, M. Waidner, and A. Weber, “The PERSEUS System Architecture,”
- [32] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, “Trust visor: Efficient TCB reduction and attestation,” *Proceedings - IEEE Symposium on Security and Privacy*, pp. 143–158, 2010.
- [33] R. Sailer, T. Jaeger, E. Valdez, R. Cáceres, R. Perez, S. Berger, J. L. Griffin, and L. Van Doorn, “Building a MAC-based security architecture for the Xen open-source hypervisor,” *Proceedings - Annual Computer Security Applications Conference, ACSAC*, vol. 2005, no. i, pp. 276–285, 2005.
- [34] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, “Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems,” *ACM SIGOPS Operating Systems Review*, 2005.
- [35] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, “SWATT: SoftWare-based ATTe-station for embedded devices,” *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2004, pp. 272–282, 2004.

- [36] A. Vasudevan, J. M. McCune, J. Newsome, A. Perrig, and L. van Doorn, “CARMA: a hardware tamper-resistant isolated execution environment on commodity x86 platforms,” pp. 48–49, 2012.
- [37] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, “SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust,”
- [38] J. Noorman, B. Preneel, K. U. Leuven, P. Agten, W. Daniels, R. Strackx, C. Huygens, F. Piessens, A. Vanherrewege, and I. Verbauwhede, “Sancus : Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base,” *22nd USENIX Security*, 2013.
- [39] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, “TrustLite,” *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*, pp. 1–14, 2014.
- [40] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, “TyTAN: Tiny trust anchor for tiny devices,” *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pp. 1–6, 2015.
- [41] S. Schulz, C. Wachsmann, and A.-r. Sadeghi, “Lightweight Remote Attestation using Physical Functions,” 2011.
- [42] A. Seshadri, M. Luk, and A. Perrig, “SAKE: Software attestation for key establishment in sensor networks,” *Ad Hoc Networks*, vol. 9, no. 6, pp. 1059–1067, 2011.
- [43] O. Girard, “OpenMSP430,” <http://opencores.org/project,openmsp430>, p. 125, 2009.

- [44] F. Kohnhäuser, A. Schaller, and S. Katzenbeiss, “PUF-Based Software Protection for Low-End Embedded Devices,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9229, pp. 315–316, 2015.
- [45] M. A. Gora, A. Maiti, and P. Schaumont, “A flexible design flow for software IP binding in FPGA,” *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 719–728, 2010.
- [46] R. Nithyanand and J. Solis, “A theoretical analysis: Physical unclonable functions and the software protection problem,” *Proceedings - IEEE CS Security and Privacy Workshops, SPW 2012*, pp. 1–11, 2012.
- [47] R. Maes, “PUF-Based Entity Identification and Authentication,” *Physically Unclonable Functions: Constructions, Properties and Applications*, vol. 9783642413, no. i, pp. 1–185, 2013.
- [48] P. Koeberl, J. Li, A. Rajan, and W. Wu, “Entropy loss in PUF-based key generation schemes: The repetition code pitfall,” *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014*, pp. 44–49, 2014.
- [49] R. Maes, P. Tuyls, and I. Verbauwhede, “A soft decision helper data algorithm for SRAM PUFs,” *IEEE International Symposium on Information Theory - Proceedings*, pp. 2101–2105, 2009.

- [50] V. Van der Leest, B. Preneel, and E. V. der Sluis, “Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment,” pp. 268–282, 2012.
- [51] S. Puchinger, M. Sven, M. Bossert, M. Hiller, and G. Sigl, “On Error Correction for Physical Unclonable Functions,” *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding; Proceedings of*, pp. 1–6, 2015.