Analysis of Microcontroller Embedded SRAMs for Applications in Physical Unclonable Functions

by

Sakib Imtiaz

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The growth of the Internet of Things (IoT) market has motivated widespread proliferation of microcontroller- (MCU) based embedded systems. Suitable due to their abundance, low cost, low power consumption and small footprint. The memory architecture typically consists of volatile memory such as block(s) of SRAM, and non-volatile memory (NVM) for code storage. Authentication and encryption safeguard these endpoints within an IoT framework, which requires storage of a secure key. Keys stored within integrated circuits (ICs) are susceptible to attack via reverse engineering of the NVM. Newer approaches use Physical Unclonable Functions (PUFs), which produce unique identifiers that takes advantage of device-level randomness induced by manufacturing process variation in silicon.

The unclonable property of PUFs is demonstrated with an analytical model. The unpredictable yet repeatable start-up values (SUVs) of SRAM bit-cells form the basis of an SRAM PUF. Performance measures, such as reliability, randomness, symmetry, and stability, dictate the quality of a PUF. Two commercial off-the-shelf (COTS) ARM-Cortex based MCU products, the STM32F429ZIT6U and ATSAMR21G18A, underwent automated and manual power cycling experiments that examined their embedded SRAM SUVs. The characterization framework provided acquires data via debug software and a developed C program, power cycling using a USB controlled relay and post-processing using Python. Applications of PUFs include cryptographic key generation, device identification and true random number hardware generation.

Statistical results and a comparative analysis are presented. Amongst the total bit-cell count of the embedded SRAM in STM and ATSAM MCUs, 36.86% and 28.86% are classified as non- or partially-skewed, respectively across $N = 10,000$ samples. The Atmel MCU outperforms the STM MCU in reliability by 1.42 %, randomness by 0.65 % and stability by 8.00 %, with a 4.74 % SUV bias towards a logic '1'. Max errors per 128-bit data item is 22 and 38 bits for MCU #1 and MCU #2, respectively. The STM MCU exhibits column-wise correlation illustrated in a heatmap, where the Atmel MCU shows a random signature. The embedded SRAM in the Atmel MCU outperforms the STM MCU's and is thereby considered the more suitable PUF.

Acknowledgements

There are several individuals who I am thankful for in my graduate studies.

I would like to express my sincere gratitude to supervisors Professor Manoj Sachdev and Dr. Derek Wright. Their expertise, resourcefulness and patience combined with the ability to maintain a professional interpersonal relationship with myself has led to an exceptional positive graduate studies experience. I am indebted to their generous acts to facilitate my learning experience as a graduate student and preparing me for the industry.

I have been fortunate to be a research member of the prestigious CMOS Design and Reliability (CDR) Group. Special thanks to my mentor Adam Neale whom was crucial in guiding me through the steep learning curve of research. Thanks to Derek for providing exceptional proof-reading and teaching me how to improve my writing ability for technical documents. Thanks to all the group members I have interacted with for sharing life stories and research endeavours – Qing, Maarten, Anthony, Morteza, Mehdi, Dhruv. In addition, I'd like to thank Professor Bruce Richmond and Professor Liang-Liang Xie for assisting me with the development of the combinatorial analytical model.

I would like to acknowledge my parents, my sister, & brother-in law & his respective family and the rest of my extended family for going above and beyond in supporting my decision to pursue graduate studies. They've been right by my side in aiding me in this journey and held a vital role in my well being.

This work has been funded by an industry partner through a NSERC Research grant for the PUF Engage Initiative.

Dedication

To my family & friends
Remembering Neelanjana. . . Fellow MASc Student

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet of Things (IoT) aims to integrate physical objects with real time computer systems connected to the Internet. This technology is made possible from broadband Internet availability, ingenuity in embedded systems equipped with microcontrollers (MCUs), and the ongoing reduction in semiconductor costs. American market research firm Gartner estimates that the IoT market currently consists of 1.6 billion devices [1]. Networking company Cisco projects this will grow to over 50 billion connected devices by 2020. [2].

1.1 Motivation

With an ever growing number of connected physical, trustworthy machine-machine communication becomes increasingly vital, and effective device authentication protocols are necessary to prevent counterfeiting [3]. These interactions bring forth a wide range of challenges including interoperability, scalability, and trustworthy communication. Encryption with a secret key is necessary for secure data transmission over public channels [4]. Secret keys stored on non-volatile memories (NVM) are susceptible to malicious reverse engineering attacks. Adversaries can exploit algorithm timing and power consumption in side channel attacks.

A promising alternative is to generate a secret key without the need to store it. In digital integrated circuits (ICs), a physical unclonable function (PUF) is the source for generating volatile, chip-specific signatures at runtime or power-on [3]. In the past decade, research indicates that PUFs may be a hopeful solution for the aforementioned security issues. The premise of this work is a study of the suitability of PUFs implemented in existing commercial off-the-shelf (COTS) products.

A PUF is an expression of an unclonable instance-specific feature of a physical object, analogous to biometric features of humans, such as fingerprints [3]. Applications of PUFs include key generation, device identification and true random number hardware generation (TRNG) [4]. There are various implementations of PUFs, including those based on IC coatings, flip-flops, latches, optical techniques, and numerous other methods [4]. Silicon PUFs, the focus of this research, use the transistor mismatches due to manufacturing process variabilities to produce a unique signature. In an SRAM-based PUF, the bit-cell start-up-values (SUVs) upon power-on form the basis of a PUF.

Embedded systems equipped with MCUs are in abundance and suitable for the IoT market due to their low cost, low power consumption and small footprint. The memory architecture in these systems typically consists of volatile memory such as small block(s) of SRAM, and some NVM for code storage. Prior to utilizing the SUVs of the SRAM bit-cells to construct PUFs, one undergoes a qualification process to determine its suitability. In this work, the analysis of embedded SRAM in COTS products − ARM Cortex based MCUs are presented in accordance to their PUF performance with regards to reliability, randomness, symmetry and stability.

In addition to existing embedded SRAM characterization work for COTS MCUs in [5] and [6], this thesis contributes a new set of reference data. Using the birthday problem [7], a combinatorial analytical model for SRAM- based PUFs is presented. Lastly, this thesis provides a PUF characterization framework for SRAMs. whether external or within a system on chip (SoC) such as a MCU.

1.2 Physical Unclonable Functions

A PUF is that it is a physical one-way function in which an input produces an output, whereas the input cannot be determined given the output. The concept of a PUF originated when Pappu observed unique speckle patterns on a transparent epoxy wafer filled with bubbles upon shining it with a laser [8]. Gassend et al. [9] introduced silicon-based PUFs. In this class of PUFs, process variation during fabrication leads to local device parameter varations used to generate chip-specific unique IDs. The unclonable property stems from this phenomenon, since the manufacturing process is stochastic in nature, resulting in random unpredictable process variations. Hardware security solutions using PUF technology are applied in IoT, embedded systems, identification, automotive, communications, content distribution, government and defense markets [2].

A PUF is a feature of a physical system that reacts to an input stimulus known as a challenge and generates a single response determined by mismatches in physical properties, together called a *challenge-response* pair (CRP). The function mapping is denoted as: $f : R_i \longleftarrow \text{PUF}(C_i, S_i)$, where:

- R_i = Corresponding output response from the instance i
- C_i = Specific input stimulus to the instance i
- S_i = Collection of all physical parameters for instance i

In Figure 1.1, an example is provided of a silicon-based PUF. The challenge and response relationship is binary and is described by bit-streams. Examples of challenges include test vectors for arbiter circuits that exercise critical delay paths, or memory addresses for SRAMs that store SUVs. A PUF with a substantial set of CRPs is classified as a *strong* PUF, otherwise classified as a *weak* PUF. A system with 1 CRP is termed as a physical obsfuscated key (POK) [3]. Strong PUFs are recommend for high security systems, limiting attackers with opportunities to decipher patterns due to the vast set of CRPs. As depicted in Figure 1.2, PUF instances return unique responses when supplied with identical challenges – the fundamental property of unclonability.

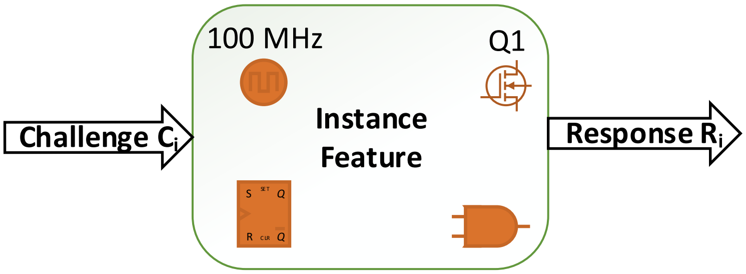


Figure 1.1: Principle of Silicon-based PUF

Based on the evaluation method, PUF primitives exist as both non-intrinsic and intrinsic forms. The coating PUF by Tuyls et al. [10] relies on measuring the capacitance between two metallic lines on an IC where the top layer is covered by an applied coating with randomly distributed dielectric particles. In this PUF along with the aforementioned optical PUF by Pappu et al. [8], the source of randomness in the system is exposed by additional equipment and hence the PUF is externally evaluated. Other PUFs under this class exist such as paper, CD, RF DNA, magnetic and accoustical. Intrinsic PUFs on the other hand rely on the inherent physical characteristics of the system, like silicon-based PUFs, and are self-evaluated. Delay-based PUFs such as arbiter, glitch and ring oscillator PUFs generate unique responses based on signal delays between critical paths [3] and oscillation frequency differences. Memory-based PUFs such as SRAM, latch, flip-flop and butterfly PUFs generate unique responses based on the probable outcome of a known state from a metastable state [4]. This thesis focuses on intrinsic or silicon based PUFs with an emphasis on SRAM-based PUF.

Properties of PUFs:

- **Evaluation:** The PUF yields a device specific output response by evaluating an input challenge. In complexity analysis, the evaluation is within one-degree polynomial time or $O(n)$, where n refers to the bit-stream length of the challenge input. This infers using minimal resources with respect to cost, area and power.

- **Unclonable:** There are no two PUFs with identical CRPs. An analytical model or

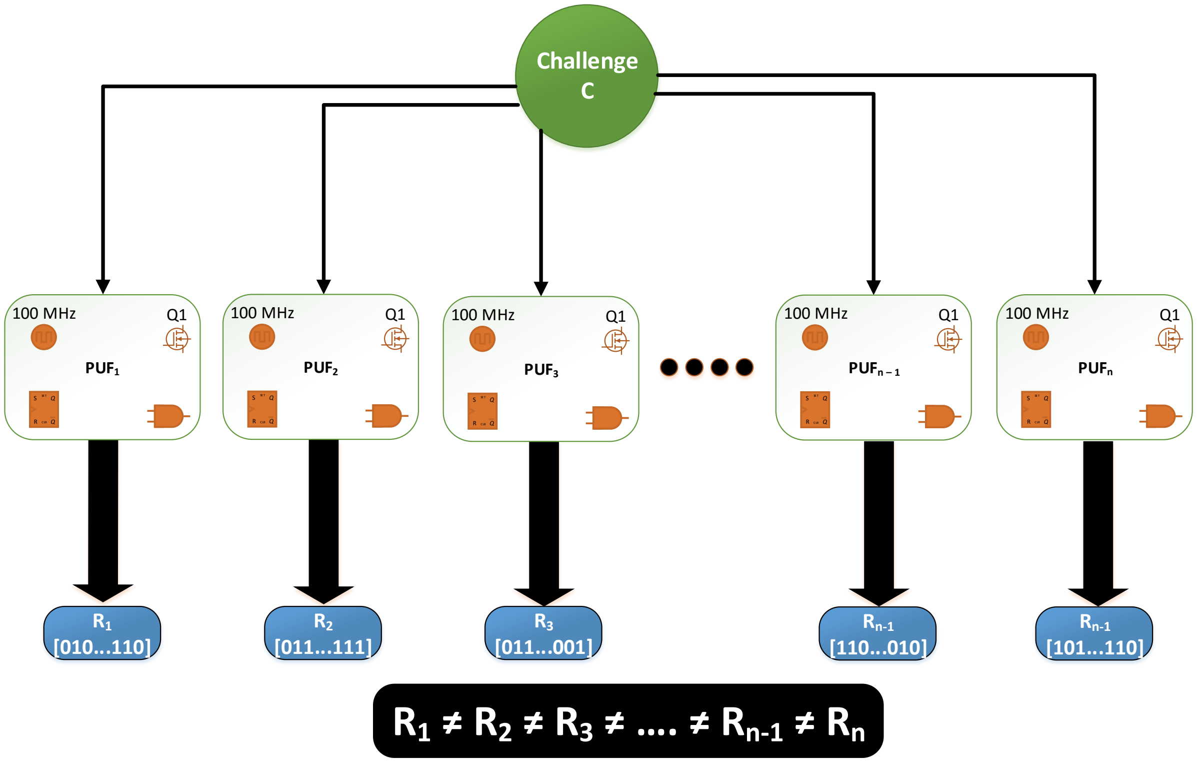


Figure 1.2: PUF Unclonable Property

probabilistic algorithms cannot be derived to predict behaviour.

- **Reproducible:** Under varied environmental conditions with respect to temperature, power supply fluctuations and aging, the PUF maintains a reliable set of CRPs.
- **Secure:** A PUF can only produce an output response given an input challenge. A PUF *cannot* provide the corresponding input challenge given an output response.

1.3 Outline

This thesis is organized as follows. Chapter 2 provides an analytical model to demonstrate the unclonable property of a PUF, a literature review on silicon based PUFs with an emphasis on SRAM memories and performance metrics to gauge PUF quality. Chapter 3 proposes a framework to characterize embedded SRAM in COTS MCUs and provides applications of PUFs. Chapter 4 entails results of empirical data acquired from embedded SRAM in two ARM-Cortex based MCUs. Chapter 5 concludes this work.

Chapter 2

Background

To understand this work, this section provides adequate background information including probability theory, characterization parameters, and SRAM memories.

2.1 Numerical Theory

This section proposes a probabilistic analytical model modelled in software (Python) to demonstrate a PUF's fundamental property, unclonability or distinctness. A Mathematician named Richard von Mises introduced the birthday problem [11] which questions how many students must be present in a classroom such that the probability that two students sharing the same birthday is 50%. Contrarily, what is the probability of two students having distinct birthdays? A combinatorial solution was derived with key assumptions:

- Number of birthdays in a year is 365 and they are equally possibly likely or evenly distributed. Probable outcome of a birth is the same across all students, or $\frac{1}{365}$.
- Anomalies, like leap years or twins are disregarded.

Using the birthday problem, an analytical model for PUFs is described. Referring to Figure 2.1, what is the probability that two PUF entities do not share the same bit-stream fingerprint, derived from a silicon based PUF primitve such as SRAM. The number of students is interchanged with the number of PUFs and the number of birthdays is interchanged with the bit-stream length which gives the total number of possible unique signatures. This analytical model is demonstrated in two forms: an uniform and non-uniform distribution. The former of which is extensively covered in this work and the latter of which is in progress. The purpose of this analytical model is to provide a theoretical means to provide design insight and understand related PUF concepts.

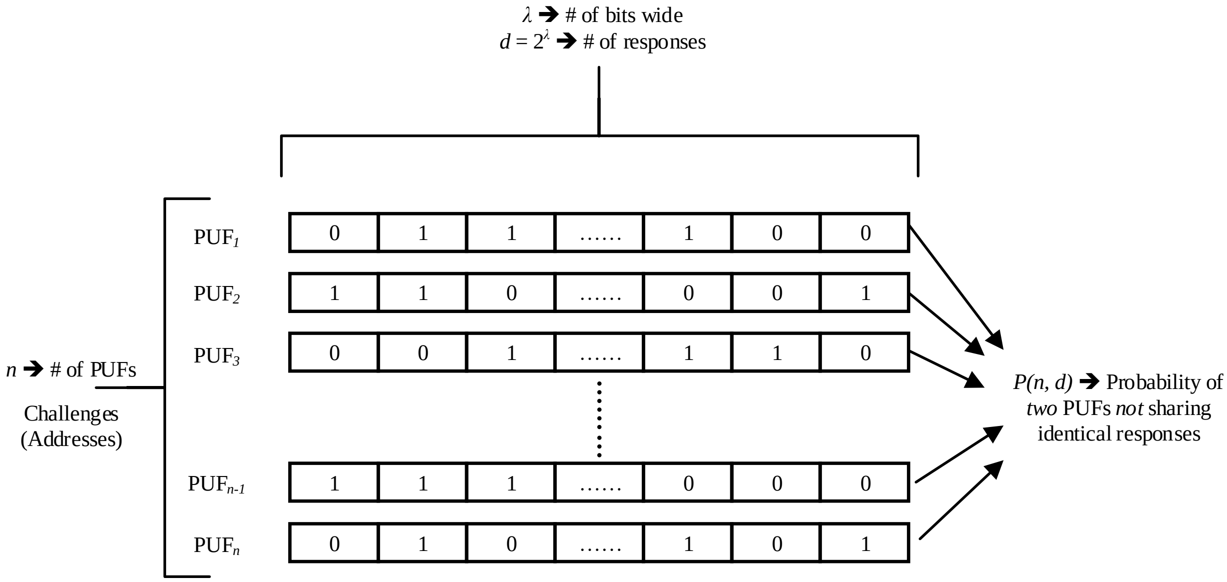


Figure 2.1: PUF Bit-stream Signature Collision Problem

Uniform Distribution

The distinctness probability $P(n, d)$ of a set of n students with d possible but equally likely birthdays is expressed in Equation 2.1. The probability that two students share the same birthday is 50% in a group of $n = 23$ with $d = 365$ birthdays [7].

$$
\begin{aligned}
P(n,d) &= (\frac{d-1}{d})(\frac{d-2}{d})\dots(\frac{d-(n-1)}{d}) \\
&= \frac{(d-1)(d-2)\dots[d-(n-1)]}{d^{n-1}} \\
&= \frac{d!}{d^n(d-n)!}
\end{aligned}
\tag{2.1}
$$

In the context of PUFs, the parameters now have the following meanings:

- $n \rightarrow$ Total number of silicon-based PUF entities
- $d \rightarrow$ Total number of unique signatures $= 2^\lambda$, where λ bit-stream length

In figure 2.2, the statistical relationship between PUF entities and bit-stream length up to $\lambda = 32$ is illustrated in log-scale. As the length λ increases, the total space of unique bit-stream signatures signatures exponentially increases. This allows for increased capacity of PUF entities in-turn reducing probability of collision.

The following observations can be made:

- For any $n > d$, $P(n, d) = 0$: There are more PUF entities than available bit-stream responses, hence at least two PUFs share identical responses.
- For $n = 1$, $P(n, d) = 1$: Since there is only PUF entity, no collisions can occur.

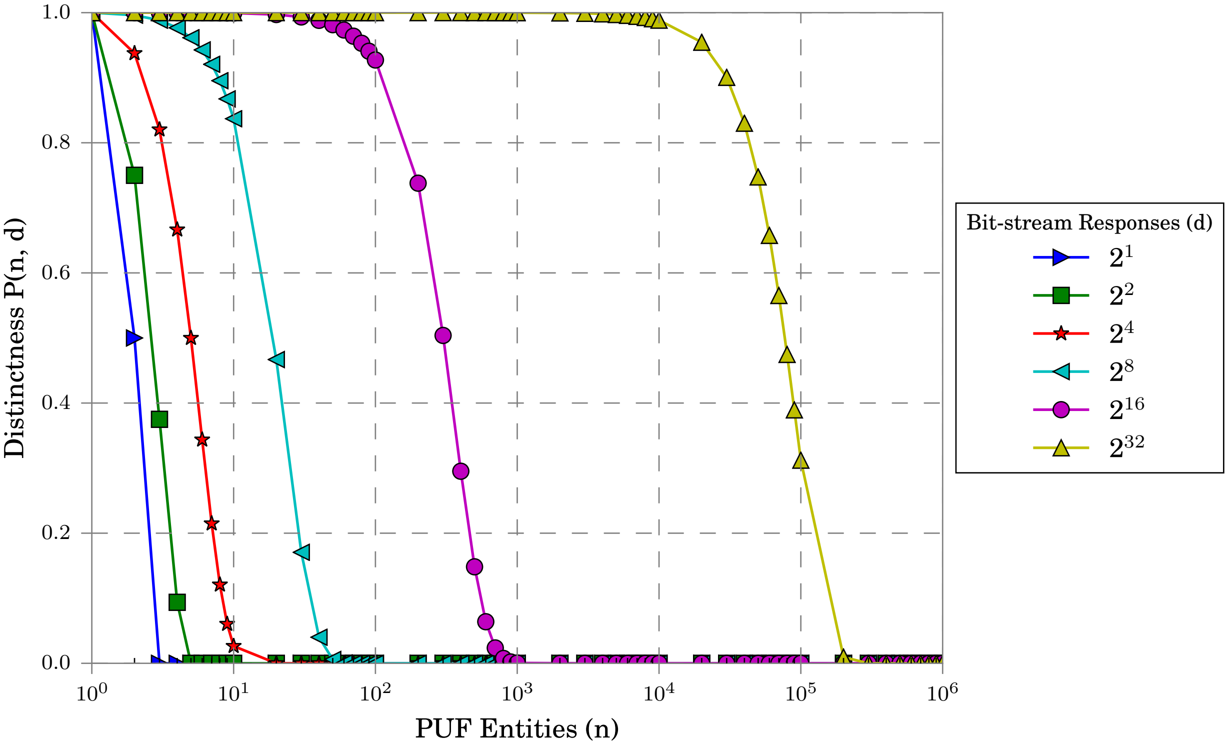


Figure 2.2: Probability of Distinctness − Exact Solution [$\lambda = 32$]

As λ grows beyond 64-bits, the exact solution becomes computationally intensive, significantly increasing program runtime. Sayrafiezadeh [7] derived an approximate solution based on the Taylor series expansion based on natural logithm base e. Equation 2.2 is used here-on to compute distinctness probabilities for $\lambda = [64, 128, 256]$. In figure 2.3, the solid line represents the actual solution and the dashed line with triangles represents the approximate solution. Deviations are noticeable for $\lambda =$ 1, 2 and *4*. Quantitatively expressed in equation 2.3 [12], the deviation error is expresses the absolute difference between the exact and approximate solution for a set of n, d parameters. As λ increases, deviation error is reduced significantly. As a sample input set of $n = 1\,x\,10^8$ and $d = 2^{64}$, the deviation error is $\epsilon \approx 4.898\,x\,10^{-16}$. Figure 2.4 illustrates distinctness up to $\lambda = 256$ using the approximate solution, similar trend as the exact solution with less runtime.

$$\begin{aligned} P(n,d) &\approx 1 - e^{\frac{-n(n-1)}{2d}} \\ &\approx 1 - (1 - \frac{n}{2d})^{n-1} \end{aligned} \tag{2.2}$$

$$\epsilon \leq \frac{n^3}{6(d-n+1)^2} \tag{2.3}$$

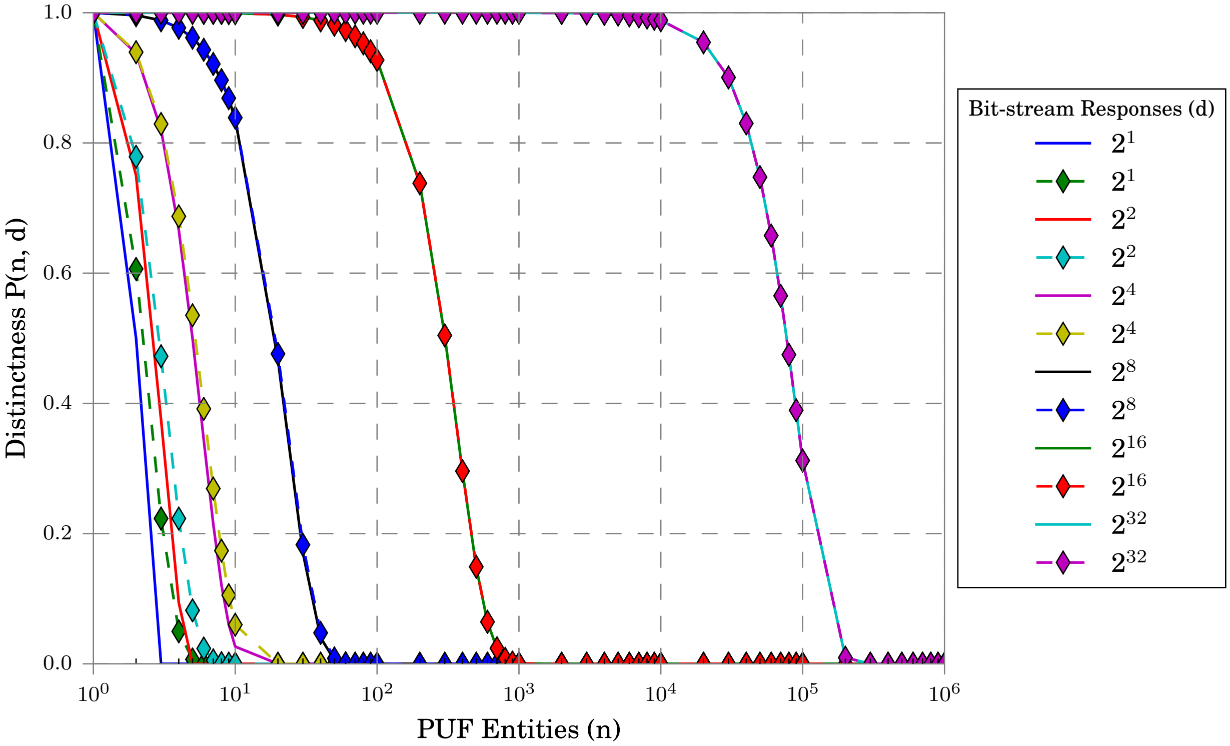


Figure 2.3: Probability of Distinctness − Exact vs. Approx Solution [$\lambda = 32$]

Table 2.1 summarizes numerical results for varied λ. Results include distinctness probability in relation to yield $P(n,d)$ with results surpassing a standard of $6\sigma = 0.999997$ [13].

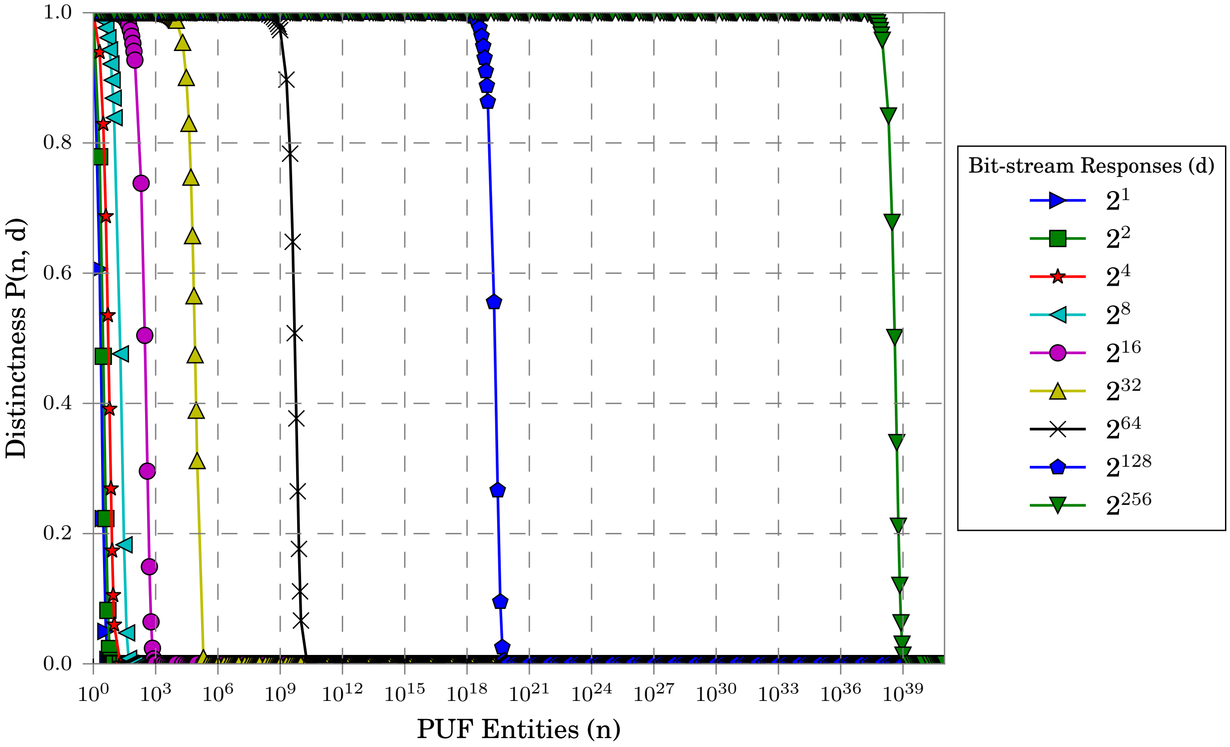


Figure 2.4: Probability of Distinctness − Taylor Series Approx [$\lambda = 256$]

Table 2.1: Summary − PUF Approximate Solution [Uniform Distribution]

Number of Signatures (2^λ)	**Collision Rate**	**Distinctness** $P(n, d)$	**PUF Entities**
65536 $\lambda = 16$	1.0e−3	0.999	10
	1.0e−4	0.9999	4
	1.0e−5	0.99999	1
	1.0e−6	0.999999	1
	1.0e−7	0.9999999	1
4294967296 $\lambda = 32$	1.0e−3	0.999	2000
	1.0e−4	0.9999	900
	1.0e−5	0.99999	200
	1.0e−6	0.999999	90
	1.0e−7	0.9999999	20
1.8446744e+19 $\lambda = 64$	1.0e−3	0.999	1.0e+8
	1.0e−4	0.9999	6.0e+7
	1.0e−5	0.99999	1.0e+7
	1.0e−6	0.999999	6.0e+6
	1.0e−7	0.9999999	1.0e+6
3.4028237e+38 $\lambda = 128$	1.0e−3	0.999	8.0e+17
	1.0e−4	0.9999	2.0e+17
	1.0e−5	0.99999	8.0e+16
	1.0e−6	0.999999	2.0e+16
	1.0e−7	0.9999999	8.0e+15
1.1579209e+77 $\lambda = 256$	1.0e−3	0.999	1.0e+21
	1.0e−4	0.9999	1.0e+21
	1.0e−5	0.99999	1.0e+21
	1.0e−6	0.999999	1.0e+21
	1.0e−7	0.9999999	1.0e+21

Non-uniform Distribution

In the work presented by Berresford [14], the birthday problem takes into consideration of an empirical data set of $239,762$ births in New York in 1977. The probability of births is non-uniformly distributed, birthdays are not equally probable. This results in a larger collision rate.

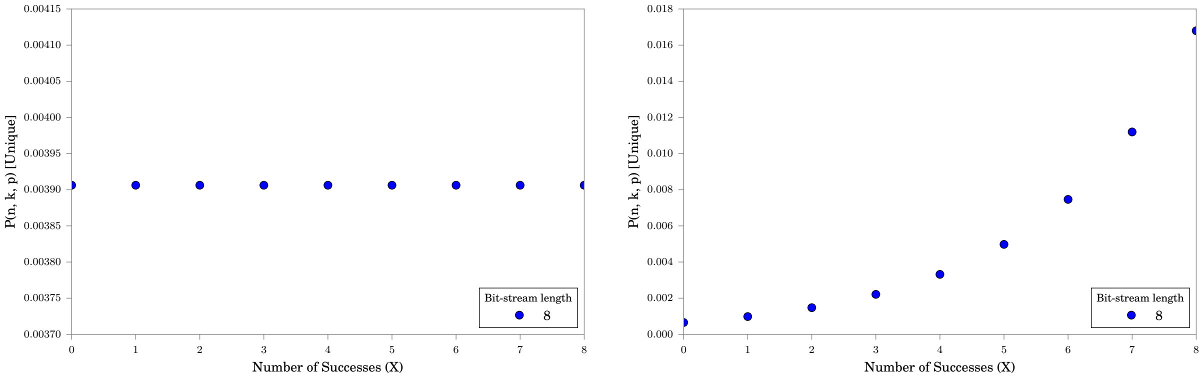
In the context of PUFs, bit-stream signatures are not always equally probable for a given λ. Individual bits in the bit-stream do not have a presumed probable outcome of $p = 0.5$ upon power-on. The bit-cell is termed *skewed*, covered at the end of this chapter. A particular bit-cell may tend to a logic '0' more often than '1' implying that for this given PUF, creating instances of it will produce signatures that comprise of more logic '0' than logic '1' resulting in increased collisions.

Consider the case of $\lambda = 2$, a total of 4 possible bit-stream responses, $[00, 01, 10, 11]$ as enlisted in Table 2.2. If each bit-cell has a probable outcome of $p = 0.5$, each signature amongst the set of all signatures is equally probable with $P = (0.5) * (0.5) = 0.25$. Presuming that we have a PUF where $p = 0.6$, implying that the bit-cell is slightly skewed towards a logic '1', and given that all PUF instances have a similar non-uniform distribution, the probable outcome of a signature 01 is $P = (1 - 0.6) * (0.6) = 0.24$.

Table 2.2: Probable Outcomes of Signatures $[\lambda = 2]$
$p_U = 0.5$ and $p_{NU} = 0.6$

p_1	p_0	$P_{Uniform}$	$P_{Non-uniform}$
0	0	$0.25 = (0.5)^2$	$0.16 = (1 - 0.6)^2$
0	1	$0.25 = (0.5)^2$	$0.24 = (1 - 0.6) * (0.6)$
1	0	$0.25 = (0.5)^2$	$0.24 = (0.6) * (1 - 0.6)$
1	1	$0.25 = (0.5)^2$	$0.36 = (0.6)^2$

A bit-stream is a series of independent Bernoulli trials $-$ where each trial results in a success (logic '1') or fail (logic '0'). The Binomial theorem expressed in Equation 2.4 accounts for non-unique signatures, as in 0110 is treated the same as 0101 due to number of successes. The Binomial coefficient $\binom{n}{k}$ is removed to account for unique signatures. Figure 2.5 portrays the unique probable outcomes of a signature for length $\lambda = 8$ versus the number of successes/logic '1's within the bit-stream. In the left Figure 2.5a, all signatures are equally probable whereas Figure 2.5b shows signatures with more logic '1's than '0's are more likely, detrimental to distinctness probability. The conventional approach used in the birthday problem utilizing a sum of remaining probable outcomes in Equation 2.1 will not be useful in this case since conditional probabilities arise.



(a) $p = 0.5$

(b) $p = 0.6$

Figure 2.5: Binomial Distribution – Unique Probable Outcomes of 8-bit Signature

$$P(n, k, p) \;=\; \binom{n}{k} p^k (1-p)^{n-k} \tag{2.4}$$

Where the binomial coefficient n choose k is mathematically expressed as:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

An initial base model can be derived based on Bernoulli trials to tackle the problem of non-uniform distributions. Given 2 PUF bit-streams:

- Find the probability that p_i is equal for PUF_1 and PUF_2 in position i. There are two cases, the i^{th} bit in both PUFs is a 1, probability p^2, or a 0, probability $(1-p)^2$, respectively. Adding these two probabilities and iterating through all λ bits (raising to the power of λ) yields probability of PUF_1 and PUF_2 being equal.

- Subtract 1 from this probability to get the probability that these 2 PUFs are distinct.

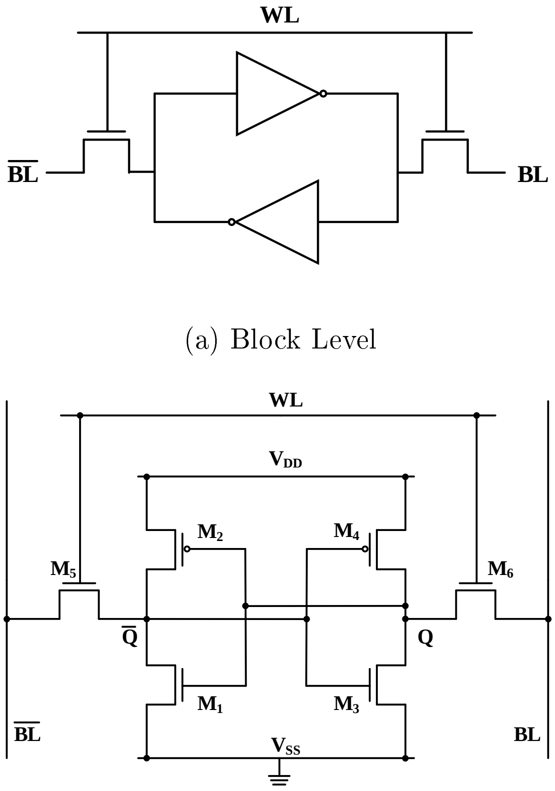
$$\begin{aligned} &\textbf{PUF 1:}\, 010\ldots101 \\ &\textbf{PUF 2:}\, 100\ldots001 \\ P_{Equal} &= [p^2 + (1-p)^2]^\lambda \\ P_{Distinct} &= 1 - P_{Equal} = 1 - [p^2 + (1-p)^2]^\lambda \end{aligned} \tag{2.5}$$

2.2 SRAM based PUFs

Static random access memory (static RAM or SRAM) is a type of semiconductor memory used in integrated circuits for a wide range of computing systems including mobile devices and enterprise servers. It's smallest functional unit, the bit-cell, is solely responsible for storing a single bit of information, a logic '0' or '1'. It is classified as volatile, where data is lost in absence of the supply voltage. A variety of topologies exist such as 6T and 8T, which represent the number of transistors to construct a functional bit-cell. In this thesis, the concept of a SRAM based PUF is demonstrated by the 6T SRAM bit-cell topology.

2.2.1 Bit-cell Architecture

Every 6T SRAM bit-cell is comprised of a fundamental storage element which is realized through CMOS technology using back-to-back inverters as illustrated in 2.6a. Surrounding this core are access devices as NMOS transistors and peripheral circuitry pertinent for read/write operations. In the low level schematic in Figure 2.6b, transistors M_1 and M_2, M_3 and M_4 form the back-to-back inverters, respectively. Transistors M_1 and M_3 are referred to herein as drivers, and M_2 and M_4 are referred to herein as loads. In this work, we focus on understanding the behaviour of SRAM bit-cells in the case of power-on.



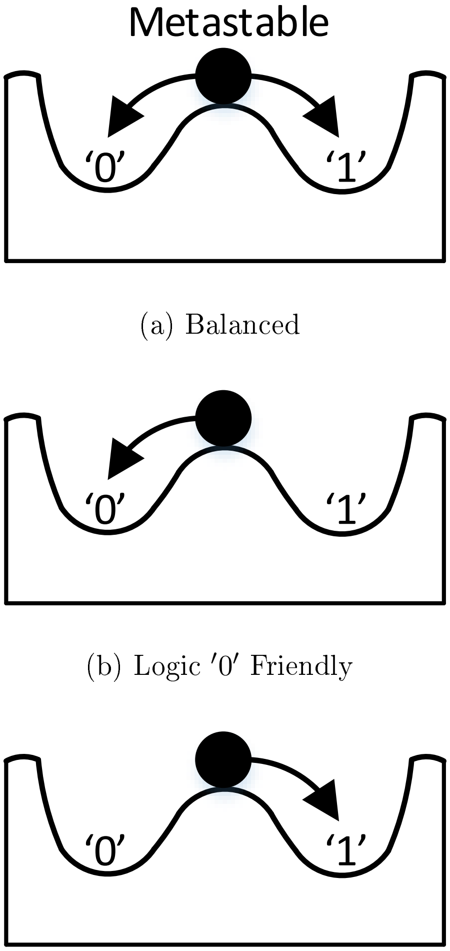
(a) Block Level

(b) Transistor Level

Figure 2.6: Schematic – SRAM Bit-cell [6T]

2.2.2 Bit-cell Start-up Behaviour

A bit-cell exhibits the behaviour of a bistable system portrayed in Figure 2.7. Ideally, a bit-cell should store a stable state upon power-up with equal probable outcomes (≈ 50% 0 or 1) from a metastable state as per illustrated in Figure 2.7a. However, process variation during fabrication leads to a potential mismatch in the storage element causing a predisposition to power on to a preferred cell state, for e.g. ≈ 71%% a logic '0' and ≈ 29%% a logic '1'. Figure 2.7b along with Figure 2.7c demonstrates this biased behaviour in a bistable system. A unique signature in the form of a bit-stream can be produced by collecting the SUVs of an uninitialized block or data word(s) of SRAM for a multitude of applications. Addressed by Kumar et. al [15], it is common for FPGA products to perform a fresh initialization sequence which involves a *hard reset* clearing all SRAM bit-cells to a logic ’0’, effectively destroying the random SUVs.



(a) Balanced

(b) Logic ′0′ Friendly

(c) Logic ′1′ Friendly

Figure 2.7: Bistable System Nomenclature

Process variation and inherent device mismatches is becoming a predominant issue as technology nodes are scaled down [16]. The mismatch in threshold voltages of the driver transistors dominates the start-up behavior [4]. Figure 2.8 shows the bit-cell’s SUV behavior when driver M_3 has a lower threshold voltage in comparison to driver M_1 with a supply voltage V_{DD} applied. In a switch-based model, M_3 turns on earlier in relation to M_1 and the node Q is pulled down to a logic '0'. With the additional load devices M_2 and M_4, regenerative feedback from the back-to-back inverters [4] stabilizes $\overline{Q}$ to a $'1'$.

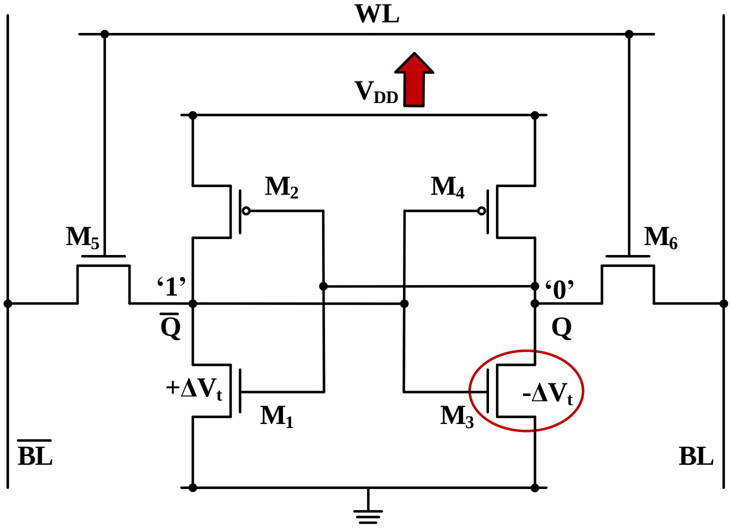
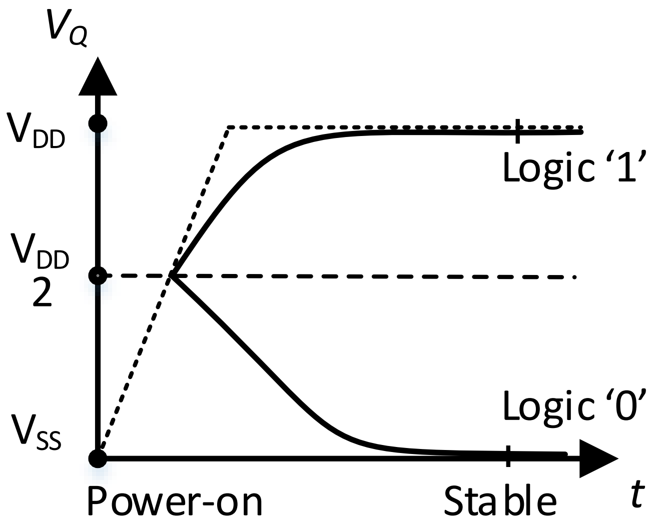
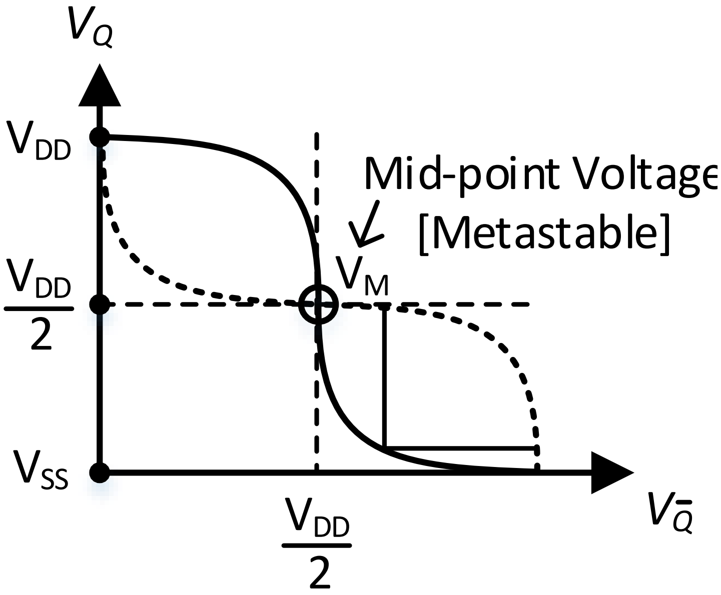


Figure 2.8: Bit-cell start-up behavior due to V_t mismatch

Electrical characteristics of SRAM bit-cells are shown in Figure 2.9. The transient response in Figure 2.9a illustrates small signal behaviour in both cases of drive strength mismatches between M_1 and M_3. In the voltage transfer characteristic (VTC) presented in Figure 2.9b, an ideal bit-cell based on a balanced set of back-to-back inverters begins off at the voltage level $\frac{V_{DD}}{2}$. Any small offset induced by mismatches is amplified and one of the two stable states is reached randomly.



(a) Transient Response of SRAM Bit-cell Start-up Behavior



(b) Static Noise Margin (SNM) Butterfly Curves

Figure 2.9: SRAM Bit-cell – Power-on Characteristic

2.2.3 Bit-cell Types

An SRAM bit-cell can be categorized under one of three types as outlined in Table 2.3. Fully-skewed cells are desired for PUF applications due to their ability to produce consistent reliable responses [17]. The symbols p_i and $q_i = (1 - p_i)$ refer to the probable outcomes of bit-cell i having logic '0' and '1' SUVs, respectively.

Bit-cell Type	Description
Non-skewed	Process variation on each half of the bit-cell cancel or nullify each other $p_i = 0.5$
Partially-skewed	Process variation causes moderate V_t mismatch, leading to a stronger driver transistor on one half of the bit-cell $0 < p_i < 1$, $p_i \neq 0.5$
Fully-skewed	Process variation causes large V_t mismatch, leading to an always preferred SUV $p_i = \{0, 1\}$

Table 2.3: SRAM Bit-cell Classification

2.3 Characterization

This section provides an overview of well-known quality metrics in the literature that demonstrate PUF performance. In subsequent equations, $a_{i,j,k}$ refers to the i^{th} bit of the j^{th} word of the k^{th} sample in PUF space a. The word size or length is λ bits, there are N words in the space, and $L = \lambda * N$ total bits in the space. The SUVs are sampled K times. A "PUF challenge" refers to an input bit-stream.

2.3.1 Reliability

This parameter is a measure of the reproducibility of the PUF. Multiple queries of the PUF with the same challenge should consistently produce the same bit-stream response. Unreliable responses (bit flips) can be tolerated by applying error correcting codes (ECC) which are stored in NVM, however introducing area and design overhead. A parameter used to characterize reliability is the fractional intra-hamming distance, $f_{HD-Intra}$, which is the number of bitwise differences between PUF responses to the same challenge normalized to the total bit-stream length M. Equation 2.6 expresses this relationship. Summing the number of bit-wise differences between samples k_1 and k_2 and normalizing with respect to L total bits yields a measure of reliability. A reference vector/golden copy such as the first response and successive responses are compared. An ideal PUF has $f_{HD-Intra} = 0$ for all pairs of samples.

$$f_{HD-Intra}(k_1, k_2) \ = \frac{1}{L} \sum_{j=0}^{N-1} \sum_{i=0}^{\lambda-1} XOR(a_{i,j,k_1}, a_{i,j,k_2})|_{k1 \neq k2} \tag{2.6}$$

2.3.2 Randomness

This parameter characterizes the uniqueness between two PUFs. There should be no correlation between bit-stream responses and they are independently random. In a given PUF instance, different challenges, should not produce correlated responses. The statistical parameter to express randomness is fractional inter-hamming distance (within-die $f_{HD-Inter}$ %). Captured in Equation 2.7, it states the number of bitwise differences between two different PUF responses, for words j_1 and j_2 within a particular sample k_0. Pairs of two responses in a PUF instance are compared. Ideally, a purely random PUF response is uncorrelated with all other PUF responses, hence the ideal $f_{HD-Inter} = 0.5$.

$$f_{HD-Inter}(j_1, j_2, k_0) \ = \frac{1}{\lambda} \sum_{i=0}^{\lambda-1} XOR(a_{i,j_1,k_0}, a_{i,j_2,k_0})|_{j1 \neq j2} \tag{2.7}$$

2.3.3 Symmetry

This parameter is concerned with the expected value of the entire SRAM PUF space. Ideally, half of the total SUVs are logic '0' and the other half are logic '1' on a single power-on. Otherwise, any systematic skew demonstrates biasing and develops an asymmetry thereby reducing the entropy of the PUF. For an SRAM PUF, the expected value is the mean, μ. Equation 2.8 shows the mean value for sample k_0. A purely uniform SRAM PUF has $\mu = 0.5$, meaning equal probabilistic outcomes of logic '0' or '1' SUVs.

$$\mu(k_0) \ = \frac{1}{L} \sum_{j=0}^{N-1} \sum_{i=0}^{\lambda-1} a_{i,j,k_0} \tag{2.8}$$

2.3.4 Stability

This parameter characterizes the stability of bit-cells within the SRAM space. Ideally, the SUVs are perfectly repeatable with all cells being fully-skewed towards '0' or '1'. Realistically, non- or partially-skewed bit-cells exist and tend to toggle their SUV response between a logic '0' and '1', vice-versa. These bit-cells exhibit random SUVs and can be disregarded from the PUF, but can still be used in hardware random number generation. The SUV probability of bit i in word j, $p_{i,j}$, can be determined with any given level of accuracy by taking a sufficiently large number of SUV samples, K, to deem whether a bit-cell be stable or not, which is assumed in Equation 2.9. Hence, a bit-cell that is for example nearly a strong '0' $p_i \ =$ or '1'. Ideally, $p_{i,j} \ = \ 1$ or 0, however non- or partially-skewed bit-cells will have intermediate probabilities. Moreover, the elimination process of these unstable bit-cells is influenced by the allowable error margin of the ECC hardware.

Unstable bit-cells in the SUV patterns can be exposed visually. Heatmaps are drawn in reference to a colour intensity scale to illustrate the various normalized bit-cell count values within the $p_i = \{0, 1\}$ spectrum. An ideal bitmap shows a random digital fingerprint with no row- or column-wise correlations. Black and white heatmaps will be illustrated for one data capture and coloured heatmaps will be used for more than one data capture. Figure

2.10 illustrates a bitmap with an ideal random response of a SRAM PUF with a grey-scale intensity expressing bit-cell probable outcome values.

$$p_{i,j} = \frac{1}{K} \sum_{k=0}^{K-1} a_{i,j,k} \tag{2.9}$$

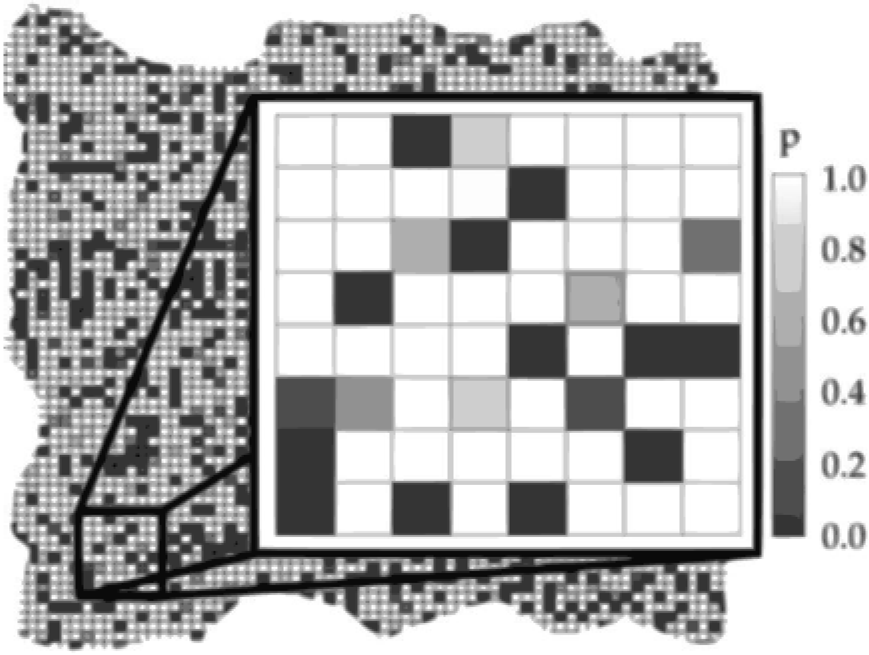


Figure 2.10: Bitmap – Ideal Response of SRAM PUF [18]

2.4 Silicon based PUFs

This section provides a literature review of silicon-based PUFs, derived from intricate process variation present in the IC fabrication processes. Sources of variability at the semiconductor level in modern CMOS technology include sub-wavelength lithography, layout-dependent transistor performance such as induced compressive stress and random dopant fluctuations (RDF) [18]. Based on their principle of operation, silicon based-PUFs are classified as follows:

- *Delay:* Using variation of interconnects in a circuit with numerous signal paths.
- *Memory:* Using device parameter mismatches in bi-stable memory storage elements.

Hereafter, existing implementations of silicon-based PUFs are discussed.

2.4.1 Delay based PUFs

Arbiter PUF

Lee et. al [19] proposed the first silicon-based PUF, the arbiter PUF. It consists of two parallel identical chains of $2-to-1$ multiplexers ending with an arbiter, a fair decision making device implemented as a SR latch recommended by Lin et. al [20] due to it's symmetrical construction. Referring to Figure 2.11, a clock edge signal transition enters the chain travelling through two unique paths and experiences propagation delay. The arbiter outputs a logic '0' or '1' response accordingly based on the race of gate delays, whether the first or second input sees the incoming clock transition signal. The challenge to this PUF system controls the select lines of the multiplexers and creates straight or crossed connections to produce CRPs. In order to exploit process variation for creating unique responses to input challenges, this necessitates the delay paths to be unbiased and perfectly symmetrical.

In the unlikely scenario of both delay paths being exactly identical for a certain input challenge, the clock transition appears at both inputs of the arbiter. The latch implementation of the arbiter will shift into a metastable state and eventually settle to an unpredictable random state based on external noise influences. This unreliability is the major drawback of arbiter PUFs. Another well known delay based PUF in the literature is the ring oscillator (RO) PUF. A chain of an odd number of inverter stages is used to implement a RO. The most common purpose of a RO is to synthesize a square waveform that serves as a clock signal in IC systems. Providing an input challenge to a number of oscillator pairs compares synthesized frequencies and thereby exposes any deviations manifested by process variation. Complex configurable RO PUFs designed by Maiti and Schaumont [21] improves reliability and randomness demonstrating progress in this PUF primitive.

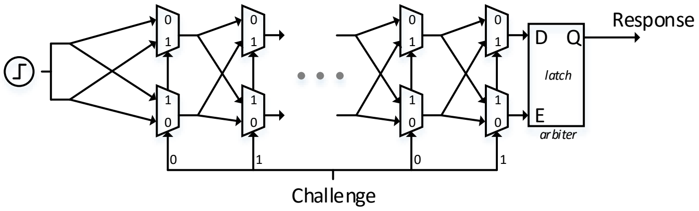


Figure 2.11: Schematic – Arbiter PUF

Glitch PUF

In a glitch PUF topology, Suzuki et. al [22] demonstrates that there is a non-linear correlation between glitch waveforms discovered in combinatorial circuits and path delays for input transitions. Whereas for arbiter PUFs, one can identify a linear correlation between the CRPs and path delays in the multiplexer chain [23]. In a combinatorial circuit, the time difference between output changes caused by a set of input transitions (challenges) can arise in intermediary unstable states at the ouput (bit-stream responses). Properties of a glitch such as occurrence and waveform shape is dictated by deviations in logical gate delay paths. As proposed in [23], the glitch PUF operation can be abstracted as follows:

- Data input transitions to a combinatorial logic circuit.
- Discovery and acquisition of glitch waveforms at the output.
- Convert glitches into response bits using hashing techniques. For e.g., perform modulo operations on the number of positive edges discovered in the glitch.

The timing diagram of a basic combinatorial block is illustrated with a *XOR* and *AND* logic gate in Figure 2.12. In the first set of inputs on the left, during the time between X_1

and X_2 both transitioning from low to high, X_3 is active beforehand and hence at the output of the *XOR* gate is logic '1' and hence the output is momentarily $Y = 1$ depicting a glitch. On the right side with the second set of inputs, X_3 is asserted during signal transitions of X_1 and X_2, however this may not produce a glitch. In the work presented by Shimizu et. al [24], an AES S-Box is used as an complex combinatorial logic block connected to a toggle flip-flop which performs even parity against the glitch count. Elegant pre-processing bit-masking techniques are used to alleviate unreliable bit-cells. As a result, a reliability of 98.7% and a randomness of 35% is reported amongst 16 FPGA boards.

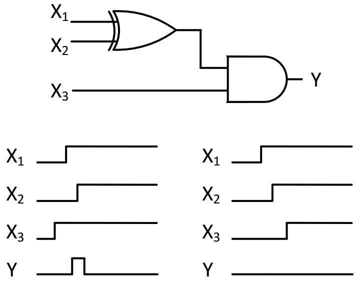


Figure 2.12: Schematic – Glitch PUF

2.4.2 Memory based PUFs

Latch PUF

The principle of operation for a latch PUF is the same as an SRAM-based PUF. Nominally matched cross-coupled or back-to-back devices exhibit random mismatches caused by process variation. In Figure 2.13, the 2 cross-coupled NOR logic gates form a SR latch. Upon power-on with the input challenge *RST* pulled to a logic '1', the output response of the PUF is stable. Once the *RST* signal is released or pulled down to logic '0', the output response shifts to a metastable state and eventually converges to a stable state, logic '0' or '1' governed by device parameter mismatches. In the proposed latch PUF implementation by Su et. al [25] using NOR gates at the 130nm node, measurements across 19-die show 96.96 % reliability, and randomness of 50.55 %.

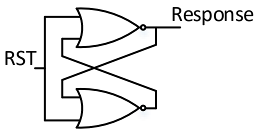


Figure 2.13: Schematic – Latch PUF

Butterfly PUF

Referring to Figure 2.14, the butterfly PUF utilizes two cross-coupled latches to retrieve an output response. Initially, the *CLR*/*Clear* signal of latch 1 and the *PRE*/*Preset* of latch 2 is driven by the input signal *Excite* causing the output response to fall into an unstable state. Once released, the output response oscillates and converges to a stable state based on physical parameter mismatches between the latches and wire interconnects. This PUF can be evaluated during power-on – unlike SRAM PUFs.

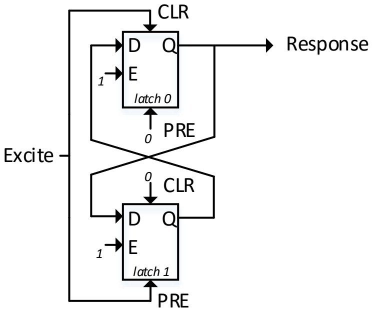


Figure 2.14: Schematic – Butterfly PUF

2.5 Applications

Commonly known useful applications of PUFs in the literature are provided in this section.

2.5.1 Secret Key Generation

Secure data transmission requires encryption of a secret key over public communication channels. Existing secure key generation is accomplished through complex TRNG hardware embedded in devices and stored in NVM – susceptible to physical or side-channel attacks. Motivated by [26], memoryless secret key storage is a use case of PUFs. Encryption algorithms such as Rivest, Shamir and Adleman (RSA) and Advanced Encryption Standard (AES) advocate a minimum key length of 3072 bits and 128, respectively [27].

In the work presented by Dodis et al. [28], and Suh and Devadas [29], a method is proposed to generate cryptographic keys. In Figure 2.15, a key generation process utilizing PUFs is illustrated. In the initialization stage, the PUF response undergoes ECC to alleviate any noisy responses caused by environment fluctuations. The ECC encoded bits or termed as *helper data* is stored in NVM or in a public database to recognize bit-flips in future PUF responses. In the regeneration stage, the key is generated using a combination of the PUF response, helper data and additional hashing techniques. Integrity of the generated key is not compromised by the publically available helper data [26].

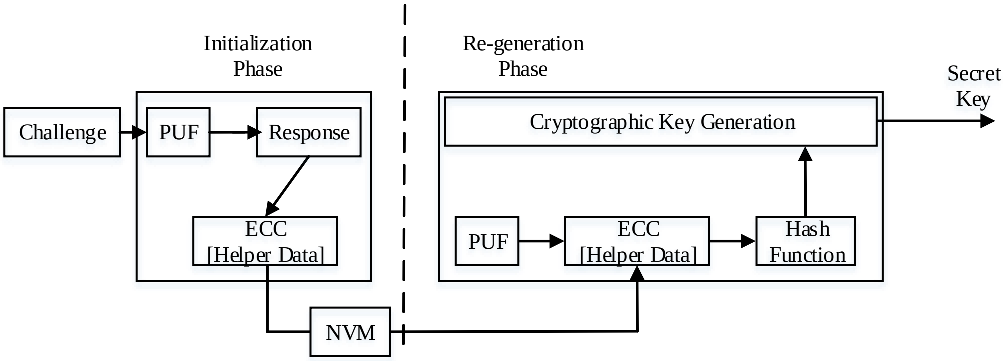
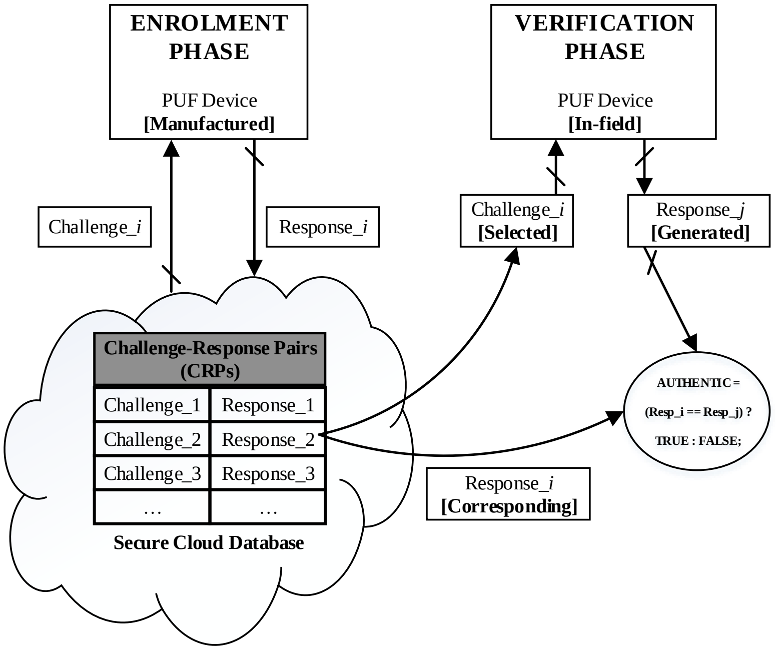


Figure 2.15: Application – Cryptographic Key Generation

2.5.2 Device Authentication

The unique output response a PUF generates is exploited for device identification and authentication. This application is divided into two steps – enrolment and verification. Referring to Figure 2.16, upon manufacturing the IC, an authentication authority enrols many, if not all, CRPs of the PUF primitive in a secure remote database. In order to verify the authenticity of the IC when it is active in the consumer market, a randomly chosen set of challenges from the database is presented to the PUF and if the generated responses and the corresponding responses match within an error tolerance described by a specification, then the PUF is successfully verified and authentic. In malicious physical attacks or attempting to clone PUFs, an error response should be encoded in addition to the PUF response authentication scheme from the perspective of the remote database. Platonov's thesis provides statistical analysis of SUVs of ten Atmel ATmega1284P MCUs [30] with proposed PUF designs for identification and key generation, results are summarized in the comparison section of Chapter 4.3.1.



Challenge-Response Pairs (CRPs)	
Challenge_1	Response_1
Challenge_2	Response_2
Challenge_3	Response_3
…	…

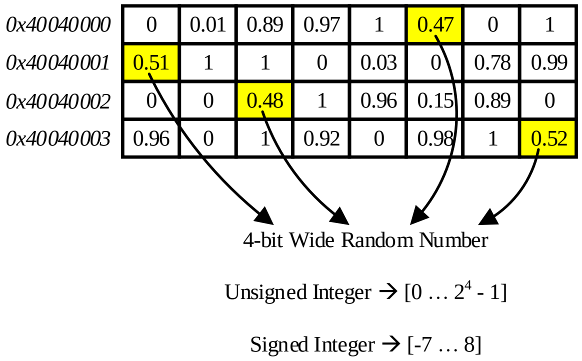
Figure 2.16: Application – Device Authentication

2.5.3 True Hardware Random Number Generation

A true hardware random number generator takes advantage of the non-skewed bit-cells in SRAM PUFs. Instead of discarding these unreliable bit-cells in the context of key generation, they can be concatenated to form a random number within a range specified by the incoming request. The overhead to identify non-skewed bit-cells can be costly based on SRAM size and a lengthy iterative power cycling process.

Figure 2.17 illustrates a small byte-addressable portion of a SRAM with an arbitrarily chosen starting address in hex and normalized probable bit-cell outcome SUVs for 100 power cycling iterations. Four non-skewed bit-cells are highlighted in yellow, $p = 0.5 \pm C$, where C is a constant adhering to a specific security requirement by the governing body Federal Processing Standards Publication (FIPS) [31]. Generated values for a 4-bit unsigned or signed integer range from $[0,\ (2^4 - 1)]$ or from $[-7,\ 8]$, inclusively and respectively. In Herrewege's PhD thesis, a lightweight PUF based TRNG is implemented using the ARM Cortex-M0 MCU [32].

For a truly random seed, maximum entropy is needed hence the random number bit generator must have a probable outcome of $p = 0.5$, hence $C = 0$. In the case of $C \neq 0$, the random bits are considered pseudo-random and the seed output is deterministic, even after applying it through a hash function. The National Institute of Standards and Technology (NIST) provides a set of comprehensive randomness tests to qualify TRNG. In the work presented by [31], the SHA-256 hash function is a suggested hash function that compresses input strings of multiples of 512 bits into an output of 256 bits. This hashing algorithm requires an input entropy of 256 truly random seed bits for it's output to have full entropy. Further conditioning algorithms utilize the output of the SHA-256 hash function in a random number generation procedure.



0x40040000	0	0.01	0.89	0.97	1	0.47	0	1
0x40040001	0.51	1	1	0	0.03	0	0.78	0.99
0x40040002	0	0	0.48	1	0.96	0.15	0.89	0
0x40040003	0.96	0	1	0.92	0	0.98	1	0.52

Figure 2.17: Application – True Random Number Generation

Chapter 3

Characterization Framework

In this section, an overview of the experimental setup is provided along with practical applications of SRAM based PUFs. Figure 3.1 depicts the principle of operation of an SRAM-based PUF. Memory information is provided as follows:

- Conventional byte-addressable direct mapped matrix with N rows by M columns.
  - Word size: $4\, bytes = 32\, bits$
  - Block size (row) = $4\, words = 16\, bytes = 128\, bits$
- Peripheral decoding circuitry for reading/writing operations.
- Knowledge of organization such as word interleaving, or parity is **unknown**. Embedded SRAM is treated as a black box DUT for PUF characterization and profiling.

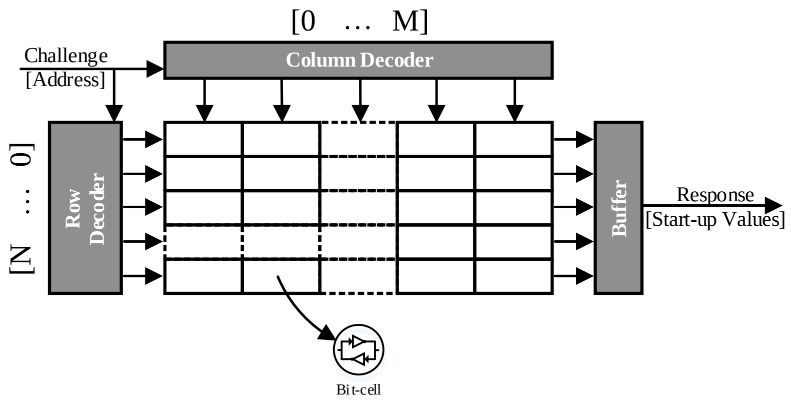


Figure 3.1: SRAM Architectural Organization

3.1 Commercial off-the-shelf Products

The commercial off-the-shelf products of interest are:

- **Microcontrollers:**
  - STM32F429ZIT6U
  - ATSAMR21G18A

3.1.1 Microcontrollers

MCUs contain a processor core, programmable input/output (I/O) peripherals, NVM for code storage and limited volatile memory typically in the form of SRAM. Majority of MCUs do not implement high performance memory architectural improvements such as hierarchy with caching mechanisms and cache coherency protocols. This allows efficient extraction of PUF responses from SUVs in the available limited SRAM due to non-intervened SRAM bit-cell content. Whereas, other programmable COTS products such as FPGAs pose difficulties to extract PUF responses due to internal resets which results in the loss of random

SUVs. Wild et. al suggest strategies using power gating and partial reconfiguration of bit-streams in FPGA development boards to extract uninitialized SRAM SUVs [33].

The COTS investigated are prototyping development boards made by silicon vendors STMElectronics and Atmel. At the core of each embedded system is an ARM Cortex-based MCU based on STM32F4 and ATSAMR21 MCU product families, respectively. They contain embedded SRAM for data storage during program execution. Table 3.1 entails product information and SRAM organization.

Table 3.1: Embedded Development Boards – Product Information

MCU	**#1**	**#2**
Manufacturer	STMElectronics	Atmel
Product [Board]	STM32F429I-DISCOVERY	ATSAMR21-XPRO
Product [MCU]	STM32F429ZIT6U	ATSAMR21G18A
Architecture	ARM Cortex–M4 180 MHz	ARM Cortex–M0+ 48 MHz
Operating Voltage	1.7 - 3.6 V	1.8 - 3.6 V
SRAM Size(s)	**Total:** 192 KB **SRAM1:** 112 KB $0x2000000 \Leftrightarrow 0x2001BFFF$ **SRAM2:** 16 KB $0x2001C000 \Leftrightarrow 0x2001FFFF$ **SRAM3:** 64 KB $0x20020000 \Leftrightarrow 0x2002FFFF$	**Total:** 32 KB $0x20020000 \Leftrightarrow 0x20008000$

3.2 Power Cycling Experiment

The experimental framework consisted of an automated and manual data acquisition strategy. Both MCU development boards received power from a 5 V USB supply voltage, supplied from a host computer. The automated strategy uses a USB programmable data relay to collect large data sets in a productive manner. The manual strategy required physical removal of the USB interface to power cycle each target board. This mimics the use case of in-field battery removal or transient operation of embedded devices.

3.2.1 Automated

A power cycling experiment, illustrated in Figure 3.2 was devised to characterize SUVs of both MCUs under a single nominal power supply condition. The embedded SRAM instance within each MCU is considered to be a black-box DUT.

Referring to Appendix A, the host machine used an automation script to perform power cycling and data transfer of SRAM SUVs. During power-on for five seconds, a supply voltage of $V_{DD-Board} = 3.3V$, derived from the USB line, $V_{DD-USB} = 5.0V$ and passing through the USB-controlled data relay, was applied to each DUT. Proprietary embedded debug software tools [34, 35] on the host machine initiated sequential SRAM reads, and data was transferred as a binary file (.bin). Power-off was for two seconds, during which time the SRAM bit-cells discharge. The power cycling repeated until the collected data set reaches $K = 10,000$ total captures.

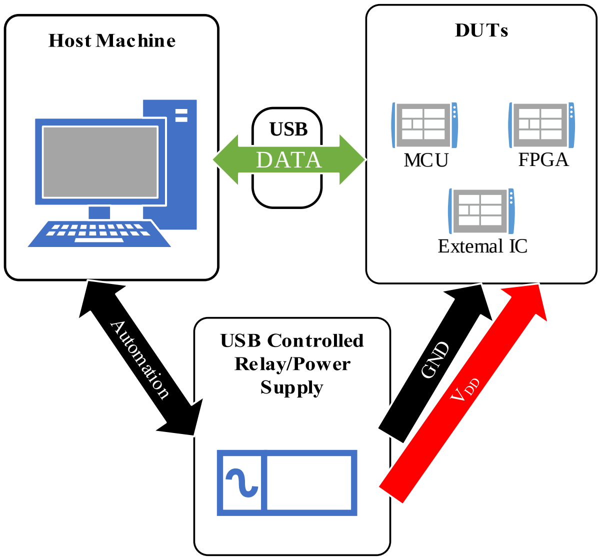


Figure 3.2: Power Cycling Experiment – Automated Setup

3.2.2 Manual

In this strategy, only the ATSAM MCU was tested to determine the benefit of performing a manual power cycling strategy. Illustrated in Figure 3.3, a console application must be open on the host machine and successfully connected to the target MCU via a COMPORT interface. In an integrated development environment, a C program is loaded onto flash memory to access and print embedded SRAM SUVs which in turn is visible on the console. Full implementation details with comments is entailed in Appendix B. This versatile API in C has been developed to facilitate access of embedded SRAM. The function prototypes, description and arguments are entailed in Table 3.2.

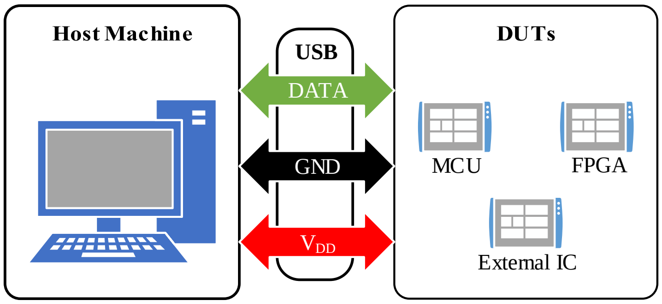


Figure 3.3: Power Cycling Experiment – Manual Setup

Table 3.2: API – Embedded SRAM Access for PUFs

Description	**Arguments**	
	Inputs	Outputs
readEmbeddedSRAM: Read SRAM contents and print byte data to terminal	**(char** $*$**)** inc_mem_start **int** inc_num_bytes	None
getPUFWord: Return 32-bit PUF data word at given address	**(char** $*$**)** inc_mem_start	**long** puf_word
getPUFLongWord: Return 128-bit PUF data word at given address	**(char** $*$**)** inc_mem_start	**long** puf_long_word

A suggested approach for implementing this API is shown in Figure 3.4. The PUF API can be integrated into the bootloader sequence at the beginning of flash memory, while the remainder is occupied by user program code. Both MCU #1 and #2 allocate the SRAM memory from the start address for stack and heap usage. The SUVs derived from source bits near the end of the SRAM address space can be utilized as PUFs.

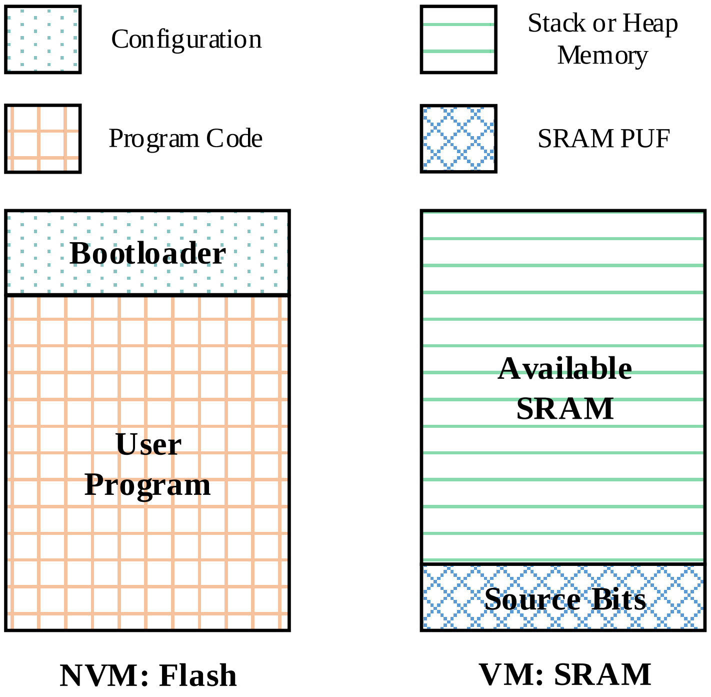


Figure 3.4: Microcontroller Memory Layout

Chapter 4

Measurement Results and Comparative Analysis

A comparative analysis between the measurement results gathered from the STM and ATSAM MCUs for both power cycling experiments is provided. The automated experiment completed under 25 hours for $K = 10,000$ samples or power cycles in an environment with an ambient temperature of 24.0 °C.

4.1 Automated

4.1.1 Reliability

The sole purpose of the automated experiment is to collect a large data set of SRAM SUVs to perform statistical analysis. The SUVs were evaluated to determine their reproducibility over $K = 10,000$ power cycles in comparison to the first captured sample. Referring to Figure 4.1, MCU #1 and #2 have calculated mean intra-hamming distances of $f_{HD-Intra} =$ 5.80 % and 4.38 %, respectively. The spikes around samples $5800, 8800$ and 9400 can be potentially attributed to temperature variations and tampering of the USB cables during data acquisition. As outlined in [36, 37], PUF reliability can be improved by applying repetition ECC using XOR logic and storing correction bits in NVM to reduce bit flips. MCU #2 performs better with respect to reliability.

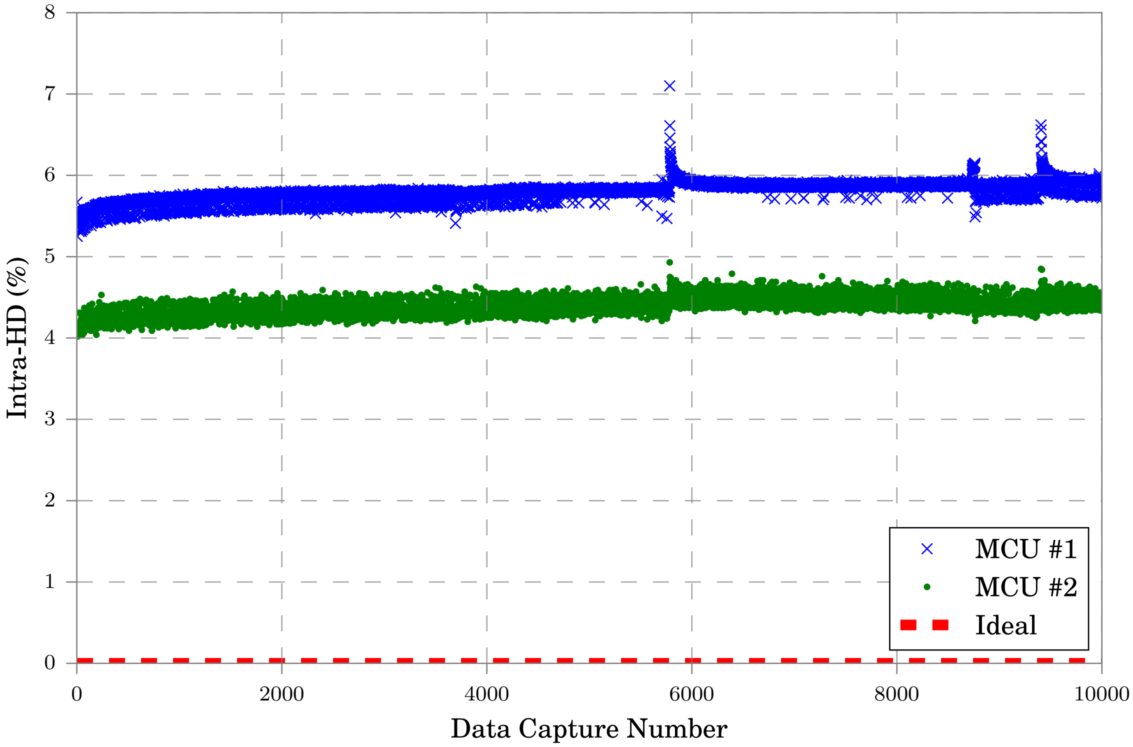


Figure 4.1: PUF Reliability – Fractional Intra-Hamming Distance [Automated] $f_{HD-Intra}$

In Figure 4.2, the number of errors per 128-bit data item are investigated. Surprisingly, despite the lower $f_{HD-Intra}$ of MCU #2, particular 128-bit data words are subject to many bit flips with a maximum $f_{HD-Intra} \approx 29.69\,\%$ implying that about 38 bits are unreliable. Whereas, for MCU #1, the maximum approaches $f_{HD-Intra} \approx 17.19\,\%$, which is about 22 bits flipped. However, on average, $f_{HD-Intra}$ for MCU #2 is about 1.223% lower than MCU #1, 4.155 % as opposed to 5.483 %, respectively. The standard deviation on the maximum errors per 128-bit data word for MCU #2 is considerably larger than MCU #1, as entailed in Table 4.1. This is important information that dictates the design complexity for resilient ECC circuitry.

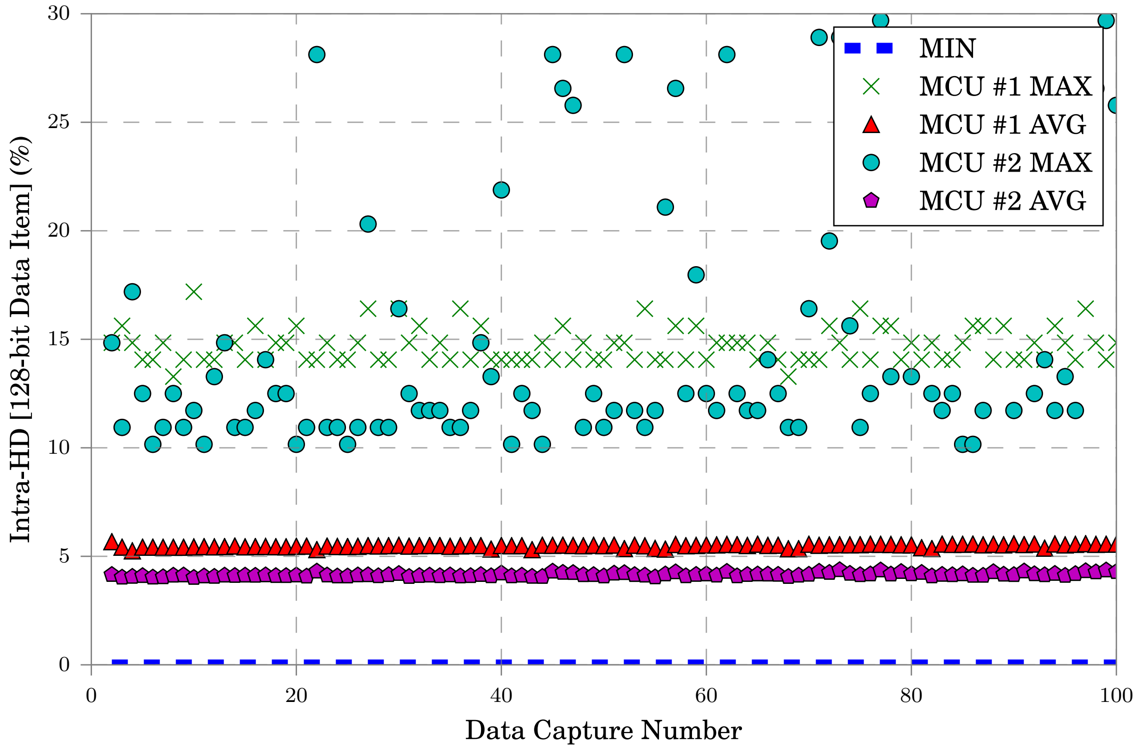


Figure 4.2: PUF Reliability − Errors per 128-bit Data Item [Automated]

Table 4.1: Summary − Errors per 128-bit Data Item [Automated | $K = 100$]

MCU	**Mean (%)**	**Max (%)**	**STD (%)**
#1	14.675	17.19	0.7798
#2	15.428	29.69	6.2690

4.1.2 Randomness

In this analysis, the PUF primitive was 128 bit data items, four aligned 32-bit words. All combinations of two distinct PUFs in the total memory space for a given sample underwent a bit-wise comparison to yield an average fractional inter-hamming distance. Referring to 4.3, MCU #1 and MCU #2 have a calculated mean within-die inter-hamming distance of $f_{HD-Inter} = 49.02\%$ and 49.67%, respectively. With respect to randomness and correlation between bit-cells in each PUF, MCU #2 marginally outperforms MCU #1.

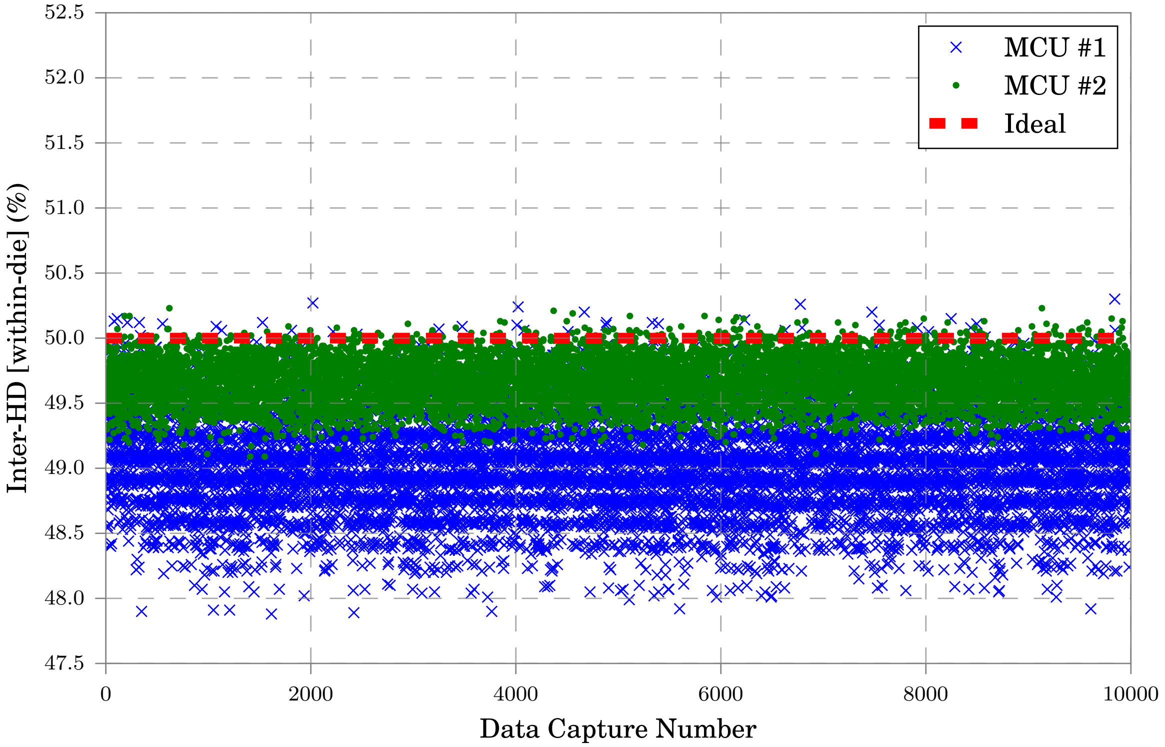


Figure 4.3: PUF Randomness – Fractional Inter-Hamming Distance [Automated] $f_{HD-Inter}$

4.1.3 Symmetry

In this analysis, the mean value of SUVs captured from one power on event was calculated. In each embedded SRAM, the PUF included the entire bit-cell population. Referring to Figure 4.4, MCU #1 has a near even distribution of SUVs with $\mu = 0.5001$. MCU #2 has $\mu = 0.5474$, the latter result means that constructed PUFs utilizing these bit-cells will have a slight bias towards logic '1', reducing PUF entropy. A summary is provided in Table 4.2 which expresses these values as percentages of '0's and '1's.

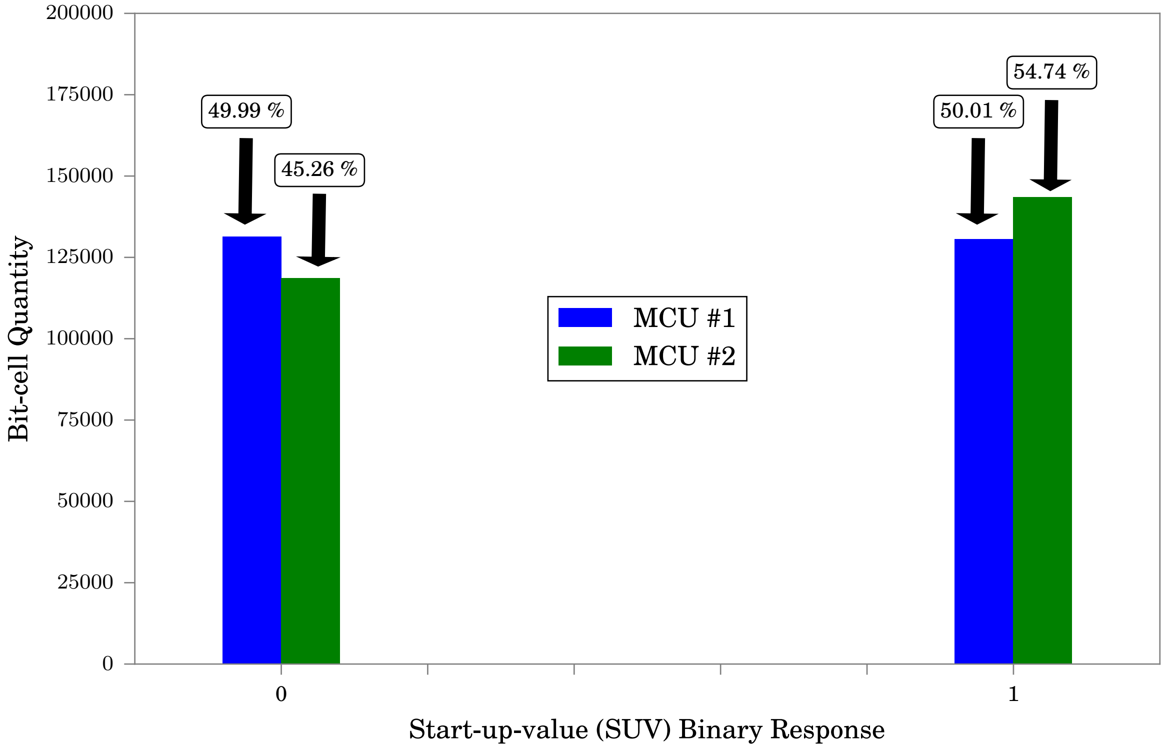


Figure 4.4: PUF Uniformity – SUV Distribution [Automated]

Table 4.2: PUF Uniformity – SUV Distribution [Automated]

MCU	**SUV Response**	**Fractional bit-cell Quantity (%)**
#1	'0'	49.99
	'1'	50.01
#2	'0'	45.26
	'1'	54.74

4.1.4 Stability

This analysis identifies unstable bit-cells using a counting technique. The sum of all 10,000 SUV samples of each unique bit-cell is calculated. If the resultant value is 0 or 10,000, then this signifies a strong '0' or strong '1', respectively. The remaining bit-cells hold values between 0 and 10,000 implying that the bit-cell SUV was erroneously '0' or '1' at least once. These bit-cells are therefore non- or partially-skewed and are not suitable for use in a PUF. Figure 4.5 shows that 36.86% and 28.86% of the total bit-cell count of MCU #1 and #2, respectively, are non- or partially-skewed. As the number of successive SUV captures increases, the number of unstable bit-cells approaches a threshold, illustrating asymptotic behaviour. The chosen number of samples 10,000 was influenced by experiment runtime in the proposed setup. Due to the linear relationship between number of samples taken and experimental runtime, if 100,000 samples were taken, the expected experimental runtime would be ≈ 250 hours. Certainly, a larger number of captures increases the confidence that a bit-cell is stable and improves the qualification process. Further, other work entailed in Table 4.7 perform experiments that capture a similar scale of data samples in the thousands.

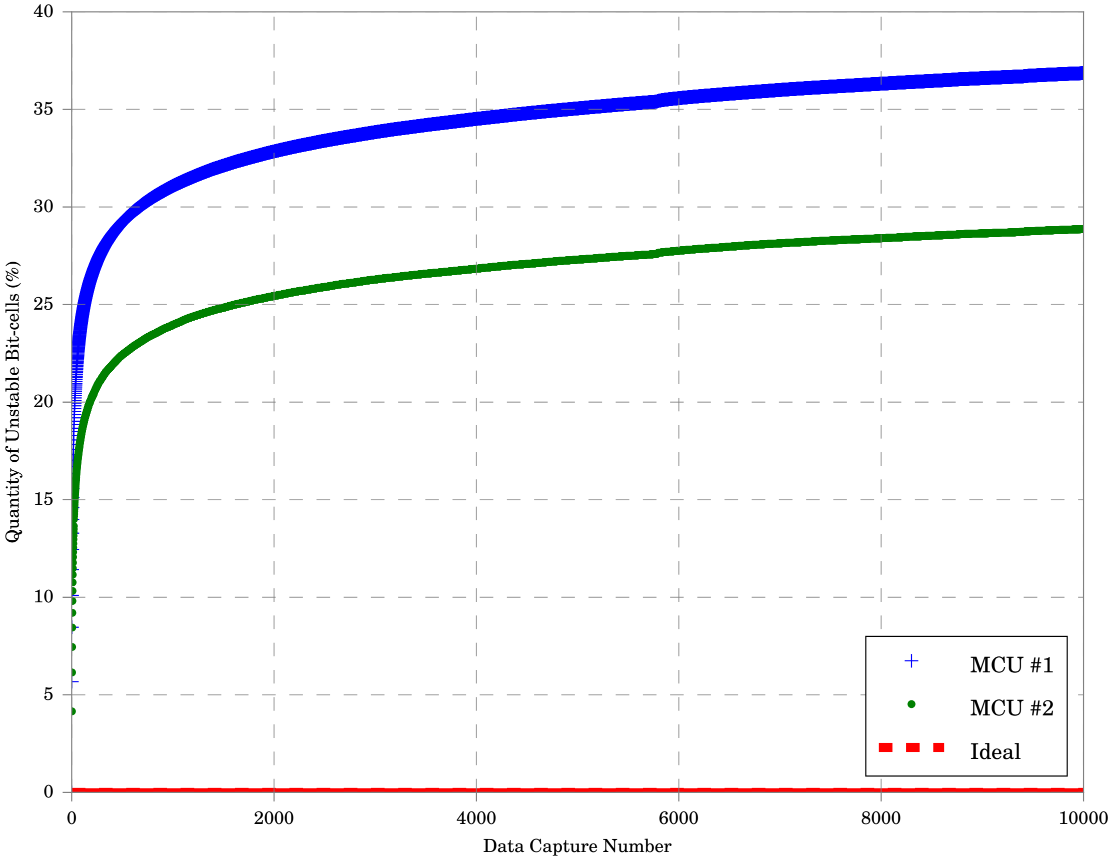


Figure 4.5: PUF Stability Unstable Bit-cells Distribution

4.1.5 Summary

In Table 4.3, reliability and randomness quality parameters are summarized. In both aspects, MCU #2 performs comparatively better and is a more suitable PUF.

Table 4.3: Summary − Reliability & Randomness [Automated | $K = 10,000$]

Metric	**MCU**	**Mean (%)**	**STD (%)**	**Max (%)**	**Min (%)**
Intra-HD	#1	5.8	0.107	7.1	5.25
	#2	4.38	0.107	4.93	4.02
Inter-HD	#1	49.02	0.359	50.3	47.88
	#2	49.67	0.172	50.23	49.09

4.2 Manual

The manual experiment involved physical removal of the USB interface to power cycle MCU #2. This conveys in-field battery removal or transient operation use cases for embedded devices housing a MCU. The goal here is to manually collect a small data set of $K = 10$ data captures and observe similar trends in comparison to analysis performed with the automated data set for MCU #2.

4.2.1 Reliability

The reliability plot showcased in Figure 4.6 has an average $f_{HD-Intra} = 4.89\%$. Recalling Equation 2.6, calculating the reliability via $f_{HD-Intra}$ requires a particular reference sample to be chosen (k_1). Results infer $f_{HD-Intra}$ is not heavily influenced by different references.

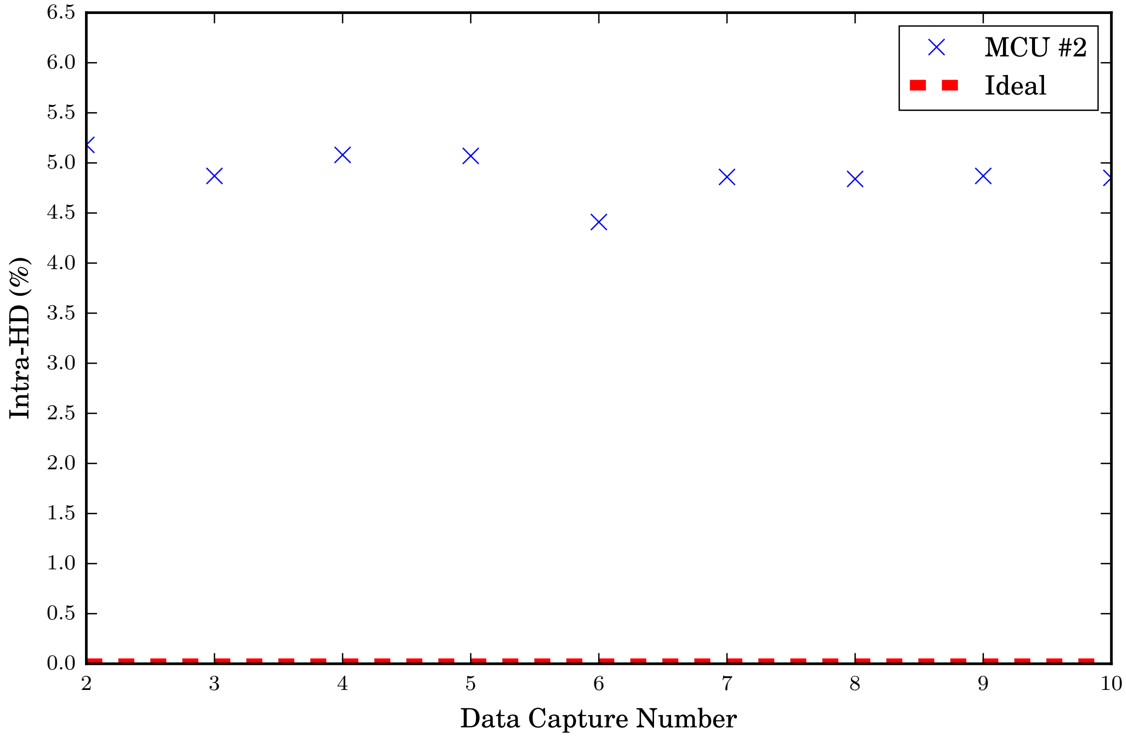


Figure 4.6: PUF Reliability Fractional Intra-Hamming Distance [Manual] $f_{HD-Intra}$

Table 4.4 shows a $\approx 0.5\%$ increase on average in $f_{HD-Intra}$ for the manual experiment in comparison to the automated experiment. This may be attributed to the physical maneuvering of the USB cable during power cycling.

4.2.2 Randomness

The randomness plot showcased in Figure 4.7 has an average $f_{HD-Inter} = 52.25\%$. This is an increase of 1.92% over the automated experiment. This is due to the pre-initialized to 0 bit-cells allocated for stack and heap memory for storing temporary data during instruction execution. Summary of $f_{HD-Intra}$ and $f_{HD-Inter}$ results are shown in Table 4.6.

Table 4.4: Reliability − MCU #2 with varying References [Manual | $K = 10$]

Metric	Experiment	Mean (%)	STD (%)	Max (%)	Min (%)
$f_{HD-Intra}$	Automated $[k_1 : 1^{st}]$	4.36	0.140	4.42	4.09
	Manual $[k_1 : 1^{st}]$	4.89	0.208	5.18	4.41
	Manual $[k_1 : 3^{rd}]$	4.83	0.060	4.92	4.71
	Manual $[k_1 : 7^{th}]$	4.88	0.147	5.18	4.73
	Manual $[k_1 : 10^{th}]$	4.91	0.105	5.09	4.71

4.2.3 Symmetry

A mean value of $\mu = 52.72$ as shown in Table 4.5, indicates a skew towards a SUV of logic '1'. This outperforms the automated experiment by a value of 2.02%. However, the allocated portion of SRAM is cleared for stack or heap memory for program code.

Table 4.5: Summary − PUF Uniformity − SUV Distribution [Manual]

MCU	SUV Response	Fractional bit-cell Quantity (%)
#2	'0'	47.28
	'1'	52.72

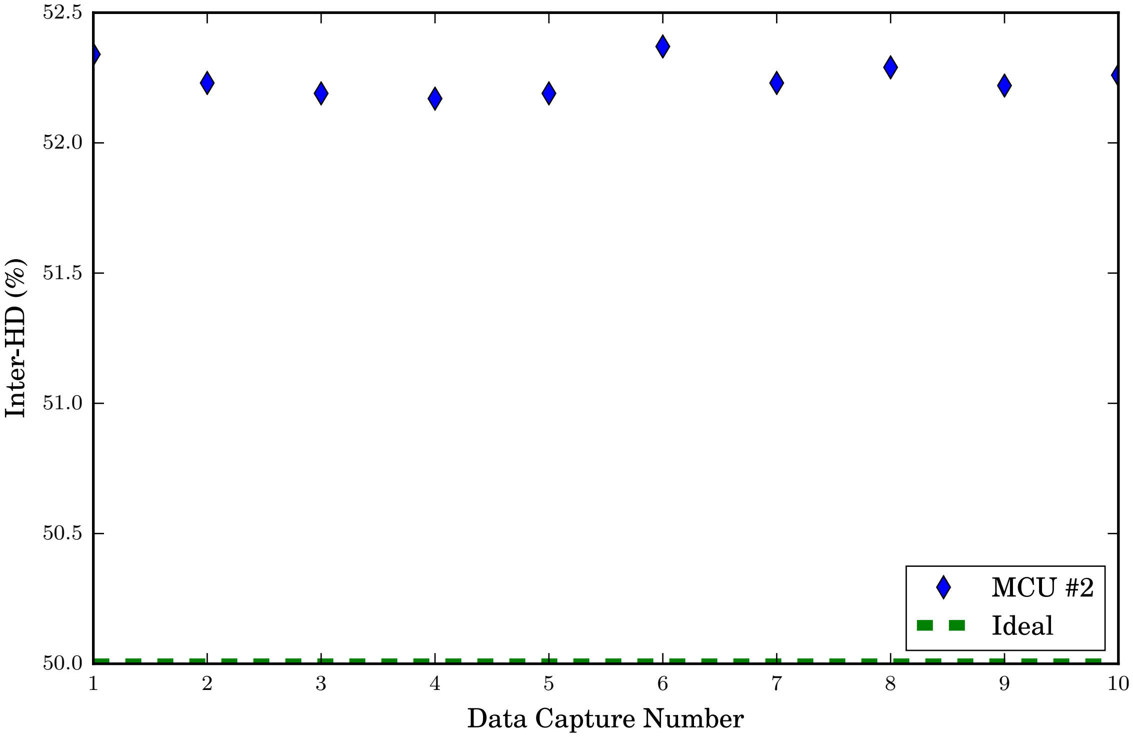


Figure 4.7: PUF Randomness Fractional Inter-Hamming Distance [Manual] $f_{HD-Inter}$

4.2.4 Stability

The sum of all 10 SUV samples of each unique bit-cell is calculated. Bit-flips from logic '0' to logic '1' or vice-versa indicate an unstable bit-cell. Figure 4.8 shows 13.30% of the total bit-cell count of ATSAM MCUs are non- or partially-skewed. A similar trend is observed in the automated experiment, with the exception that reserved bit-cells at the beginning of the SRAM space are wiped to 0, they are discarded from the unstable bit-cell distribution.

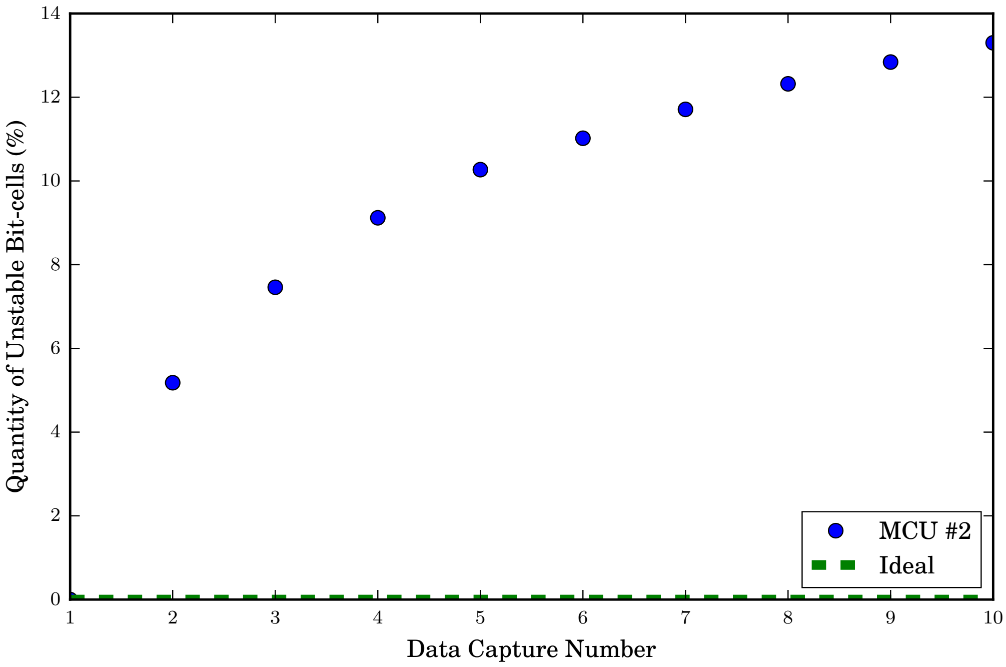


Figure 4.8: PUF Stability Unstable Bit-cells Distribution [Manual]

4.2.5 Summary

Reliability and randomness metrics are summarized for the manual experiment in Table 4.6. Automated power cycling is recommended during the characterization of SRAM PUFs.

Table 4.6: Summary − Reliability & Randomness for MCU #2 [Manual | $K = 10$]

Metric	**Mean (%)**	**STD (%)**	**Max (%)**	**Min (%)**
Intra-HD	4.89	0.208	5.18	4.41
Inter-HD	52.25	0.063	52.37	52.17

4.3 Profiles

In this section, SRAM heat-maps are drawn to illustrate bit-cell stability, based on the data set acquired through the automated experiment. Grey and colour scales are used for one data capture, and more than one data capture, respectively using Matplotlib 2D graphics library [38]. Intensity ranges from $\{0, 1\}$ in which p_i maps to based on normalized bit-cell count values to total number of samples. The start and endpoints of the colour bar indicate a strong '0' or '1' with intermediary values as unstable bits.

Illustrated by streaking patterns in Figure 4.9 and 4.10, the heatmap of MCU #1 exhibits column-wise correlation. Despite MCU #1's promising numerical results, quantitative performance metrics cannot solely describe the quality of a PUF. One can speculate that the physical organization of memory is word interleaved, where consecutive memory addresses are allocated to each bank of memory. The translation scheme from logical to physical addresses is unknown making it challenging to determine the root cause of this pattern.

Referring to Figure 4.11 and 4.12, the bitmap of MCU #2 appears random. This resembles an ideal bitmap previously shown in Figure 2.10 with high entropy SUVs.

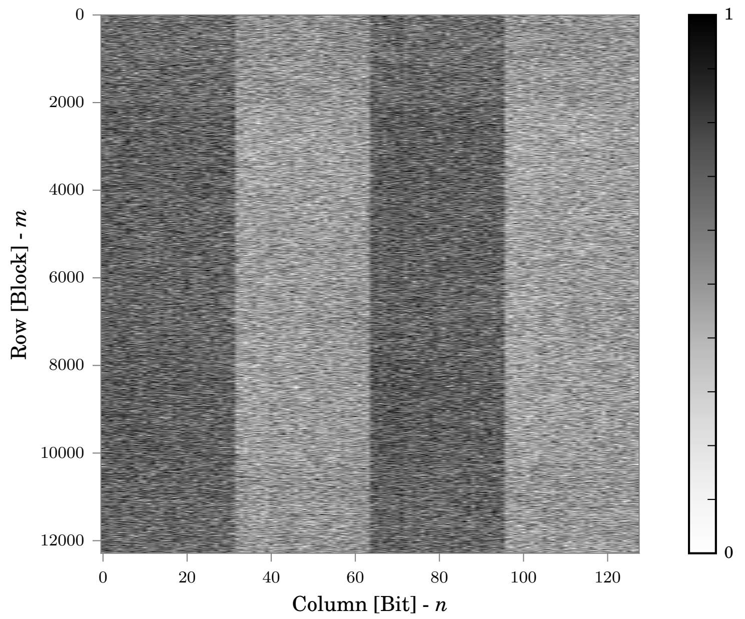


Figure 4.9: Memory Profile Heatmap – MCU #1 $[K = 1]$

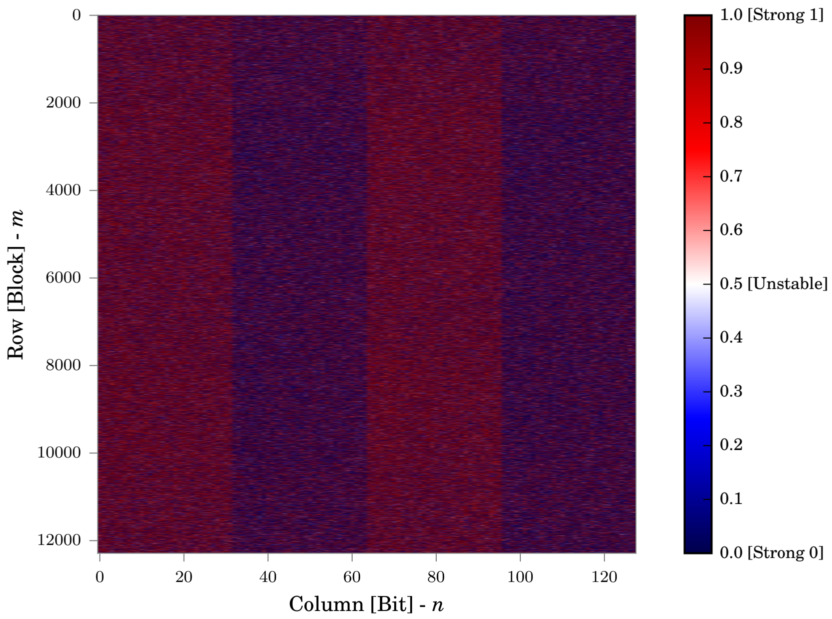


Figure 4.10: Memory Profile Heatmap – MCU #1 $[K = 10,000]$

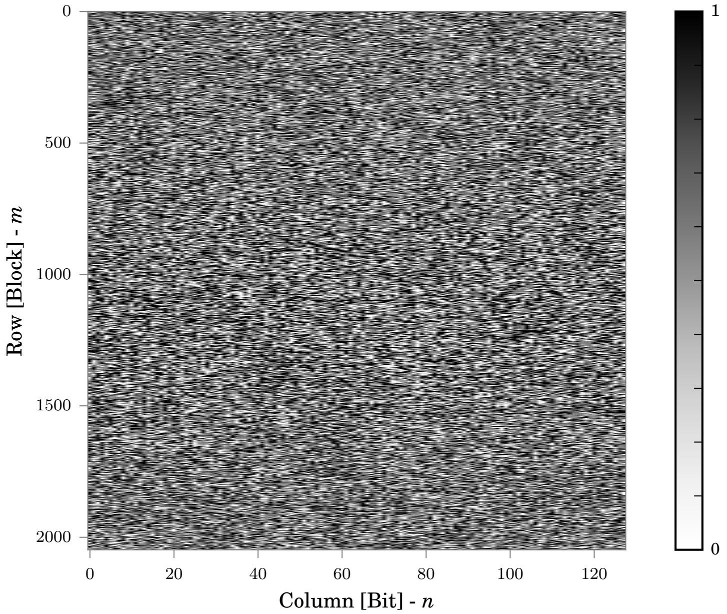


Figure 4.11: Memory Profile Heatmap – MCU #2 [$K = 1$]

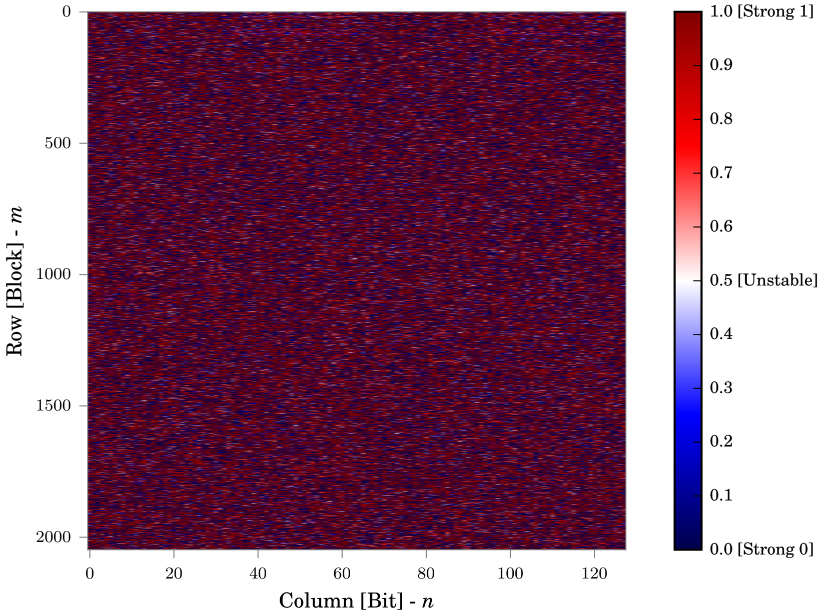


Figure 4.12: Memory Profile Heatmap – MCU #2 [$K = 10,000$]

4.3.1 Comparison

In this section, MCU #2 is chosen to be compared against other works with regards to the aforementioned quality parameters for various COTS microcontroller embedded SRAMs. The automated data set statistical results are used. As outlined in Table 4.7, MCU #2 performs well in relation to the other MCUs, in particular a relatively low $f_{HD-Intra}$ and close to ideal $f_{HD-Inter}$. Whereas, the ATmega1284P in Platonov's thesis [30] has poor $f_{HD-Inter}$ and the NXP LPC1768 has nearly twice as much $f_{HD-Intra}$. Our MCU does suffer from a bias towards logic '1', which is not evident in other works. The comparison does not adhere to a standard experimental setup, as each work has a unique test-bed for power cycling and data acquisition. Further, a varied number of MCUs are tested with different temperatures, SRAM sizes and number of captured samples.

Table 4.7: Summary – Comparison of Microcontroller Embedded SRAMs

	This Work	**[5]**	**[30]**	**[36]**
Year	2016	2015	2013	2011
Product	ATSAMR21G18A	STM32F3/F4	ATmega1284P	NXP LPC1768
Chips Tested	1	67	10	3
SRAM (KB)	32	112	16	32
Temp (°C)	24	25	20	25
Samples	10,000	2331	'Thousands'	1000
Intra-HD (%)	4.38	3.036	6.7	8
Inter-HD (%)	49.67 [within-die]	48.428 [within-die]	40 [die-to-die]	49.78 [die-to-die]
Symmetry (μ)	0.5474	0.4955	0.73	0.5205
Instability (%)	28.86	*N/A*	39	*N/A*

Chapter 5

Conclusion

This section concludes the thesis with a summary, an overview of results and future work.

Motivated by the IoT and low-cost appeal of MCUs, this thesis' intent was to determine the suitable PUF amongst two COTS Arm Cortex MCUs. The unclonability property of PUFs is demonstrated with an analytical model. Evolution of PUFs and related background information regarding PUF types, SRAM bit-cells and characterization are provided. An embedded SRAM characterization framework is developed and used in an automated power cycling experiment to collect SUV data. A manual experiment is also used for data acquisition and uses application code implemented in C. Statistical analysis of SUV measurements describe PUF performance in accordance to quality parameters.

Results show that MCU #2 outperforms MCU #1 in reliability by 1.42 %, randomness by 0.65 % and stability by 8.00 %, with a 4.74 % SUV bias towards a logic '1'. Max errors per 128-bit data item is 22 and 38 bits for MCU #1 and MCU #2, respectively. Heat-map of MCU #2 illustrates a random signature whereas MCU #1 shows column-wise correlation. Comparative analysis shows that MCU #2 is the suitable PUF.

Future work involves expanding the qualification process to incorporate advanced entropy metrics, investigating die-to-die correlations, temperature sensitivity and various power supply strategies. An investigation in the column-wise correlations of the STM MCU is suggested.

Bibliography

[1] Gartner. "Smart Cities Will Use 1.6 Billion Connected Things in 2016". http://www.gartner.com/newsroom/id/3175418. Accessed: 2016-06-08. 1

[2] Cisco. "New Cisco Internet of Things (IoT) System Provides a Foundation for the Transformation of Industries". https://newsroom.cisco.com/press-release-content?articleId=1667560. Accessed: 2016-06-09. 1, 3

[3] Roel Maes. *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer, Berlin, Germany, 2013. 1, 2, 3, 4

[4] Christopher B. and Maximillian H. *Physical Unclonable Functions in Theory and Practice*. Springer-Verlag, New York, USA, 2013. 1, 2, 4, 18, 19

[5] M. Barbareschi, E. Battista, A. Mazzeo, and N. Mazzcocca. "Testing 90nm Microcontroller SRAM PUF Quality". In *10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, Naples, Italy, April 21-23, 2015. IEEE. 2, 55

[6] G. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. de Groot, V. van der Leest, G.J. Schrijen, M. van Hulst, and P. Tuyls. "Evaluation of 90nm 6T-SRAM as Physical Unclonable Function for secure key generation in wireless sensor nodes". In *IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 567–570, Rio de Janeiro, Brazil, May 15-18, 2011. IEEE. 2

[7] Wolfram MathWorld. "Birthday Problem". http://mathworld.wolfram.com/BirthdayProblem.html. Accessed: 2016-06-10. 2, 9, 10

[8] P. S. Ravikanth. *"Physical One-Way Functions"*. PhD thesis, Massachusetts Institute of Technology, 2011. 3, 4

[9] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. "Silicon physical random functions". In V. Atluri, editor, *Proceedings of the 9th ACM conference on Computer and communications security (CCS)*, pages 148–160, New York, USA, Nov. 18, 2002. ACM. 3

[10] P. Tuyls, G.J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. *Read-Proof Hardware from Protective Coatings*, pages 369–383. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 4

[11] Teacher Link. "The Birthday Problem". `http://www.teacherlink.org/content/math/interactive/probability/lessonplans/birthday/home.html`. Accessed: 2016-06-20. 7

[12] M. Sayrafiezadeh. The birthday problem revisited. *"Mathematics Magazine"*, 67(3):220–223, 1994. 10

[13] Tutorials Point. "Six Sigma - Measure Phase". `http://www.tutorialspoint.com/six_sigma/six_sigma_measure_phase.htm`. Accessed: 2016-07-09. 11

[14] G.C. Berresford. "The uniformity assumption in the birthday problem". *Math Magazine*, 53(5), Nov. 1980. 14

[15] S. S. Kumar, J. Guajardo, R. Maes, G.J. Schrijen, and P. Tuyls. "The butterfly PUF protecting IP on every FPGA". In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 67–70, Anaheim, CA, USA, June 9, 2008. IEEE. 18

[16] N. H. E. Weste and D. M. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective 4th Edition.* Pearson, Boston, Massachusetts, 4 edition, March 11, 2010. 18

[17] A. Dargar, M. Cortez, S. Hamdioui, and G.J. Scchrijen. "Modeling SRAM Start-up Behavior for Physical Unclonable Functions". In *Proceedings of the 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, Washington, D.C., USA, Oct. 3-5, 2012. IEEE Computer Society. 21

[18] D. Holcomb. "Randomness in Integrated Circuits with Applications in Device Identification and Random Number Generation". Master's thesis, University of Massachusetts Amherst, 2014. x, 24, 25

[19] J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. van Dijk, and S. Devdas. "A technique to build a secret key in integrated circuits for identification and authentication applications". In *VLSI Circuits, 2004. Digest of Technical Papers*, pages 176–179. IEEE, 2004. 25

[20] L. Lin, D. Holcomb, D. K. Krishnappa, P. Shabadi, and W. Burleson. "Low-power sub-threshold design of secure physical unclonable functions". In *ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, ISLPED '10, pages 43–48, Austin, Texas, USA, Aug. 18-20, 2010. IEEE. 25

[21] A. Maiti and P. Schaumont. "Improving the quality of a Physical Unclonable Function using configurable Ring Oscillators". In *International Conference on Field Programmable Logic and Applications*, pages 703–707, Prague, CZ, Aug. 31 – Sept. 2, 2009. IEEE. 26

[22] D. Suzuki and K. Shimizu. "The Glitch PUF: A New Delay-PUF Architecture Exploiting Glitch Shapes". In *CHES Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems*, pages 366–382, Santa Barbara, CA, USA, Aug. 17, 2010. Springer-Verlag Berlin. 26

[23] R. Silwal. "Asynchronous Physical Unclonable Function using FPGA-based Self-Timed Ring Oscillator". Master's thesis, University of Toledo, August 2013. 26

[24] K. Shimizu, D. Suzuki, and T. Kasuya. "Glitch PUF: Extracting Information from Usually Unwanted Glitches",. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, 95-A(1):223–233, Dec. 2011. 27

[25] Y. Su, J. Holleman, and B. P. Otis. "A Digital 1.6 pJ/bit Chip Identification Circuit Using Process Variations",. *IEEE Journal of Solid-State Circuits*, 43(1):69–77, Jan. 28, 2008. 27

[26] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi, and Pim Tuyls. "Efficient Helper Data Key Extractor on FPGAs". In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008: 10th International Workshop, Washington, D.C., USA*, pages 181–197, Berlin, Heidelberg, Aug. 10 -13, 2008. Springer Berlin Heidelberg. 29

[27] Boxcryptor. "AES and RSA Encryption". `https://www.boxcryptor.com/en/encryption`. Accessed: 2016-07-10. 29

[28] Y. Dodis, L. Reyzin, and A. Smith. "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data". In C. Cachin and J.L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings*, pages 523–540, Berlin, Heidelberg, May 2-6, 2004. Springer Berlin Heidelberg. 29

[29] G.E. Suh and S. Devadas. "Physical Unclonable Functions for Device Authentication and Secret Key Generation". In *Proceedings of the 44th Annual Design Automation conference, San Diego, CA, USA*, pages 9–14. IEEE, June 4 - 8, 2007. 29

[30] M. Platonov. "SRAM-Based Physical Unclonable Function on an Atmel ATmega Microcontroller". Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Department of Computer Systems, May 1, 2013. 30, 55

[31] Vincent van der Leest, Erik van der Sluis, Geert-Jan Schrijen, Pim Tuyls, and Helena Handschuh. *Efficient Implementation of True Random Number Generator Based on*

SRAM PUFs, pages 300–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 31

[32] A. V. Herrewege. *"Lightweight PUF-based Key and Random Number Generation"*. PhD thesis, KU Leuven - Arenberg Doctoral School, January 2015. 31

[33] A. Wild and T. Güneysu. "Enabling SRAM-PUFs on Xilinx FPGAs". In *24th International Conference on Field Programmable Logic and Applications (FPL), Munich, GE*, pages 1–4. IEEE, Sept. 2 - 4, 2014. 35

[34] STM. "STM32 ST-LINK Utility". `http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link004.html`. Accessed: 2015-08-19. 36

[35] Alex Taradov. "CMSIS-DAP programmer (Formerly Atmel EDBG programmer)". `https://github.com/ataradov/edbg`. Accessed: 2015-08-17. 36

[36] C. Böhm, M. Hofer, and W. Pribyl. "A Microcontroller SRAM-PUF". In *5th International Network and System Security (NSS), Milan, Italy*, pages 269–273. IEEE, Sept. 6 - 8, 2011. 40, 55

[37] A. Neale and M. Sachdev. "A low energy SRAM-based physically unclonable function primitive in 28 nm CMOS". In *Custom Integrated Circuits Conference (CICC), San Jose, USA*, pages 1–4. IEEE, Sept. 28 - 30, 2015. 40

[38] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. 52

Appendix A

Power Cycling Automation Script

This batch script ran on a Windows host machine via command line. It interfaced with debug software to perform sequential reads in embedded SRAM for both MCUs. Data captures are in the form of binary files and a a USB controlled relay is used for power cycling. Note that directories need to be modified accordingly to the respective file system.

```
@ECHO off
setlocal enabledelayedexpansion

::Clear console log
CLS

::Memory capture commands
::Atmel -> "C:\Users\Sakib\Documents\Atmel Studio\
6.2\edbg-master-working\edbg" -brf <file_name>

::ST -> "C:\Program Files (x86)\STMicroelectronics\STM32 ST-LINK Utility\
ST-LINK Utility\ST-LINK_CLI" -Dump 0x20000000 0x30000 <file_name>

SET utility_atmel="C:\Users\Sakib\Documents\Atmel Studio\
6.2\edbg-master-working\edbg"
SET utility_st="C:\Program Files (x86)\STMicroelectronics
\STM32 ST-LINK Utility\ST-LINK Utility\ST-LINK_CLI"
SET utility_usb="C:\Program Files (x86)
\DenkoviRelayCommandLineTool\DenkoviRelayCommandLineTool_10.jar"
SET utility_usb_ser_num=DAE002dM
SET date=12_22

SET /a capture_total=10000
SET /a capture_number=1

SET folder_dest_atmel="../ATSAMR21G18A/captures_%date%_10000_samples_automated"
SET folder_dest_st="../STM32F429ZIT6U/captures_%date%_10000_samples_automated"

::Create ST directory if it doesn't already exist
IF EXIST %folder_dest_st% ECHO ST Directory already exists
IF NOT EXIST %folder_dest_st% MKDIR %folder_dest_st%

::Create Atmel directory if it doesn't already exist
IF EXIST %folder_dest_atmel% ECHO Atmel Directory already exists
IF NOT EXIST %folder_dest_atmel% MKDIR %folder_dest_atmel%

ECHO "Ensuring all individual relays are off"
java -jar %utility_usb% %utility_usb_ser_num% 4 turn 0000
ECHO ""
TIMEOUT 1

:LOOP
ECHO "Current Capture number is %capture_number%"
```

```
SET file_name_atmel="%date%_MCU_ATSAMR21G18A_32KB_%capture_number%.bin"
SET file_name_st="%date%_MCU_STM32F429ZIT6U_192KB_%capture_number%.bin"
SET file_name_absolute_atmel="%folder_dest_atmel%/%file_name_atmel%"
SET file_name_absolute_st="%folder_dest_st%/%file_name_st%"

ECHO "Turning on 1st and 2nd relay - check status LEDs on board"
java -jar %utility_usb% %utility_usb_ser_num% 4 turn 1100

ECHO "Ensuring devices are fully on - PC recognition sound"
TIMEOUT 5

::Both Atmel & ST Devices should be recognized by PC via USB port
::ST SRAM Capture
ECHO ST MCU: Capturing and saving %file_name_absolute_st%
%utility_st% -Dump 0x20000000 0x30000 %file_name_absolute_st%

::Atmel SRAM Capture
ECHO Atmel MCU: Capturing and saving %file_name_absolute_atmel%
%utility_atmel% -brf %file_name_absolute_atmel%

TIMEOUT 1

java -jar %utility_usb% %utility_usb_ser_num% 4 turn 0000

TIMEOUT 1

IF EXIST %file_name_absolute_st% (
    IF EXIST %file_name_absolute_atmel% (
        SET /a capture_number=capture_number+1
    )
) ELSE (
    ECHO "Error: Could not capture file %file_name_absolute_st%"
    ECHO "Error: Could not capture file %file_name_absolute_atmel%"
)

IF %capture_number% LEQ %capture_total% GOTO loop
```

Appendix B

Embedded SRAM Access API Code

```
void readEmbeddedSRAM(char * inc_mem_start, int inc_num_bytes) {
        // Point to starting physical address of SRAM
        char *mem_start = inc_mem_start;

        // Specify number of bytes to be read
        int num_bytes = inc_num_bytes;

        // Bit-stream field initialization to zero
        long puf_word = 0x00000000;

        // Iterate through address space until all bytes read
        for (int i = 0; i < inc_num_bytes; i = i + 4) {
                puf_word = getPUFWord(mem_start, i);

                // Print 32-bit PUF word to screen
                printf("ADDRESS   :   DATA");
                printf("%p : %x \n", (int*) (mem_start + i), puf);
        }
}
```

```
long getPUFWord(char * inc_mem_start, int inc_start_idx) {
        // Construct 32-bit PUF using bit-shifting and
        // concatenation of 1-byte (8-bit) data items

        char puf_byte_one = inc_mem_start[inc_start_idx];
        char puf_byte_two = inc_mem_start[inc_start_idx + 1];
        char puf_byte_three = inc_mem_start[inc_start_idx + 2];
        char puf_byte_four = inc_mem_start[inc_start_idx + 3];
        long puf_word = (long) (puf_byte_four | (puf_byte_three << 8)
                        | (puf_byte_two << 16) | (puf_byte_one << 24));

        return puf_word;
}

long * getPUFLongWord(char * inc_mem_start) {
        // Acquire four separate 32-bit PUF words
        long puf_word_one = getPUFWord(inc_mem_start);
        long puf_word_two = getPUFWord(inc_mem_start + 4);
        long puf_word_three = getPUFWord(inc_mem_start + 8);
        long puf_word_four = getPUFWord(inc_mem_start + 12);

        // Concatenate four 32-bit PUF words to
        // construct a 128-bit PUF long word
        long puf_long_word[4] = {puf_word_one, puf_word_two, puf_word_three,
                                 puf_word_four};

        // Aux: Printing purposes
        printf("ADDRESS: %x => %x %x %x %x \r\n", mem_start, puf_long_word[0],
                        puf_long_word[1], puf_long_word[2], puf_long_word[3]);
        return puf_long_word;
}
```