Authentication Algorithms modelling and Simulations of an Arbiter PUF

by

Vaseem Khan

Rajeev Gandhi Proudyogiki Vishwavidyalaya, Bhopal M.P., India, 2007

A Report Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Authentication & Metastability Analysis of an Arbiter PUF

by

Vaseem Khan

Rajeev Gandhi Proudyogiki Vishwavidyalaya, Bhopal M.P., India, 2007

SUPERVISORY COMMITTEE

Dr. Fayez Gebali, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Haytham El Miligi, Supervisor

(Department of Electrical and Computer Engineering)

ABSTRACT

Physical attacks represent a threat to intellectual property, confidential data, and service security because they typically involve reading and modifying data. Attackers frequently have access to tools and resources that can be utilised, either invasively or non-invasively, to read or corrupt memory. Secret keys for cryptographic techniques are often kept in memory. Physical Unclonable Functions (PUFs), which dynamically construct keys only when necessary and do not need to be retained on a powered-off chip, appear to be a potential remedy for such issues.

PUFs are circuit primitives that use inherent differences of microchips made during the manufacturing process to produce distinctive "fingerprint" output sequences (response) to a particular input (challenge). The PUF is a fantastic choice for creating cryptographic keys since these modifications are stochastic, device-specific, hard to duplicate even by the same manufacturer using similar procedures, tools, and settings, and are intended to be static.

The delay based PUF, an arbiter PUF, is the subject of our study. It benefits from the differences in propagation delays that are present between two symmetrical channels. Without the need for helper data or secure sketch techniques, we created some of the most modern algorithms that may be used to enable solid authentication and secret key generation. Finally, we present data that demonstrates how these devices behave and how their functionality is influenced by the chosen authentication mechanism and key system variables.

Contents

LIST OF TABLES

LIST OF FIGURES

Acknowledgment

I am grateful to Allah, the Most Merciful, as well as to my parents, spouse, and friends for their unflagging encouragement, wishes, and support. I would want to express my gratitude to my supervisor, Dr. Fayez Gebali, for all his assistance, direction, mentoring, flexibility, and encouragement during this program. I am also appreciative of the University of Victoria for providing a conducive learning environment.

Dedication

I dedicate this work to my beloved father Mr. Haleem Khan who always motivated me for pursuing my engineering career.

CHAPTER 1

INTRODUCTION

This chapter provides a brief introduction about Arbiter PUF. Services, private information, and intellectual property are all currently threatened; hence, cryptographic techniques are utilised to safeguard them. Cryptographic algorithms are a set of mathematical rules that encrypt or decrypt data using cryptographic keys [1]. The keys are frequently kept in non-volatile memory, like flash or EEPROM. As a result, the level of cryptographic security depends on how difficult it would be for an adversary to read the memory. Physical memory assaults, such as side-channel attacks, can be carried out by adversaries having physical access [2]. Integrated circuits (ICs) can be physically tampered with using methods like laser cutting, micro-probing, and glitch attacks to get digitalized cryptographic keys[3].

The limits of non-volatile memory key storage have been addressed by the concept of physical unclonable functions (PUFs) [4]. The PUF is a feature of a physical system. Even if an attacker has complete access to the device, PUFs use natural manufacturing process irregularities to create device-specific keys that cannot be copied or stolen. The device does not save the secret keys; they are only generated when needed. The chip or IC's tamper resistance is increased as a result.

A delay based PUF [6] is offered as an arbiter PUF [5]. Two delay channels and one arbitrator are the components of a basic arbiter PUF. The arbiter is located at the conclusion of the routes. In Fig. 2.1, the fundamental design of an arbiter PUF is depicted.

Each row in the system will feature S 2x2 switching stages and an edge-triggered flip-flop (DFF) operating as the output arbiter. The system will have W rows. A response is a B-bit choice from one of the W rows and a challenge word is made up of S bits from c[0] to c[S - 1]. The setup-hold time specifies the minimum amount of time that the input signal needs to be stable before and after the clock edge for proper operation of the circuit. Tsetup should be larger than the time difference between the upper and lower output pulses from the switching stage at position S-1.The setup time requirement specifies that the input signal must be stable for a minimum duration before the clock edge arrives. In order to meet this requirement, the setup time (Tsetup) needs to be greater than the time difference between the upper and lower output pulses from the switching stage at position S-1. This ensures that the input signal has settled before it is captured by the circuit. If Tsetup is smaller than the time difference, it would violate the setup time requirement and may lead to unreliable data capture or errors.

Since the data input to the RS flip-flop or D-type FF is the upper output of the switch stage S-1, metastability is one of the major issues with arbiter PUF. The D-FF is clocked using the lower output. In a perfect world, the rising edges of these two signals would arrive at or very close to the same moment. This will violate the setup and hold timing restrictions. Therefore, it is to be expected that several of the response bits will be undetermined or noisy.

1.1 Motivation

One of the biggest difficulties facing the Internet of Things (IoT) right now as it starts to appear in our daily lives and potential industrial systems is cyber security. IoT devices must overcome a number of difficult obstacles, including low energy usage, a shortage of processing resources, and the requirement to protect devices from cyberattacks. Unfortunately, the cryptographic algorithms that can be implemented on these devices are limited by energy footprint issues as well as a lack of processing resources. Traditional security measures like an asymmetric handshake, which calls for hashes and asymmetric cryptography, are made challenging to implement as a result. Early attacks demonstrate how frequently vendors are unaware of how simple it is to attack their goods, as is the case when they let remote updates on an unsecured Telnet port. At the same time, IoT devices must often be inexpensive, increasing the challenge to implement security functions for them.

The security triangle in IoT has three sides: authentication, authorization, and privacy. The initial defence against a cyberattack is authentication. PUFs, or physical unclonable functions, have been suggested as a simple, affordable, and widespread solution. PUFs are particularly intriguing for IoT devices with limited resources since, crucially for IoT developers, they offer to enable absolutely safe authentication without any cryptographic assets on the device. This motivates to develop an algorithm which clearly authenticates the valid devices at the field and differentiates the valid and fake devices.

1.2 Report Organization

The report's structure is as follows.

Chapter 1 presented the problem and offered a high-level summary of the project. The project's associated work and motivation were described, as well as the report's format. Chapter 2 details the (Physical Unclonable Function) PUF's background, application, threats, defence, and evaluation. Chapter 3 presents the Arbiter PUF's statistical model, response encoding and Authentication algorithms. Chapter 4 gives the performance evaluation of Arbiter PUF for all the Authentication algorithms. Chapter 5 provides conclusions for the performance evaluation.

CHAPTER 2

BACKGROUND

This chapter provides details of Physical Unclonable Function (PUF), applications of PUF, threats on PUF and Evaluation of PUF. A Physical Unclonable Function (PUF) is a physical object that is mapped on a physical device and can be used to generate completely random integers. A PUF output is straightforward to analyse, but forecasting one is far more challenging. It makes use of the distinctiveness of the physical microstructure, and even if the devices are made the same way, the PUF are unlikely to provide the same results [9]. During manufacturing, unpredictable and uncontrollable random physical factors introduce the particular attribute of microstructure.

2.1. Physical Unclonable Functions

Physical unclonable functions (PUFs), which are electronic circuits or components that produce unique outputs based on the physical properties of the device. Each PUF is unique because the physical properties of the device vary from one instance to another, even when manufactured under the same conditions. The unique output of a PUF can be used as a source of randomness or to generate cryptographic keys.

In Physical Unclonable Functions, an input called a challenge is utilised to produce an output called a response. A challenge-response pair (CRP) is a security mechanism that involves a device or system generating a unique response to a challenge presented to it. The challenge is typically a random or unpredictable value, while the response is a value that is computed or derived from the internal state of the device or system.

2.2. Delay-Based PUF

A well-researched PUF called delay-based collects response bits from an integrated circuit (IC) or embedded device via propagation delay. A PUF based on IC needs to comply with specific conditions in order to be considered intrinsic, such as:

1. The IC/embedded system must fully incorporate the PUF design and measuring unit.

2. All of the primitives in the PUF should be made possible by the fabrication of the IC or embedding device.

The challenge and response pairs are therefore not meant to leave the IC if the prerequisites are met, and the entire PUF manufacturing process should be free of overhead expenses, i.e., no extra components should be manufactured.

2.2.1. Arbiter PUF

The Arbiter PUF was initially presented [7,8,9], and it is the most frequently studied. Several improvements have been suggested, all of which improved the challenge-response pairings' statistical outcomes. The initial suggested architecture's main concept was to establish two symmetric channels with identical delay and let two signals race along them. If the two paths are symmetric, the variance of the chip manufacturing process determines the outcome of the race, which is essentially random. The physical characteristics of the material, which define the precise latency of the path, are altered during manufacture, making the material device dependent. Figure 2.1 basic representation of the APUF's architecture consists of serially coupled Flip-Flops/Multiplexers with an arbiter at the very end.

A rising edge signal is applied to the first mux's input to start the race. The arbiter then evaluates whether signal arrived first and generates a response bit of 1 or 0, depending on whether the path will be crosslinked or straight. It's important to keep in mind that each c-bit challenge results in a one-bit answer, which makes modelling assaults simpler.
On a chip, there are slight delay variations between the symmetric pathways. The arbiter PUF creates a single bit at the circuit's output end by taking advantage of the inherent delay variations between two symmetrical routes [10]. At the end of the chain, there is one arbiter and numerous switch blocks.

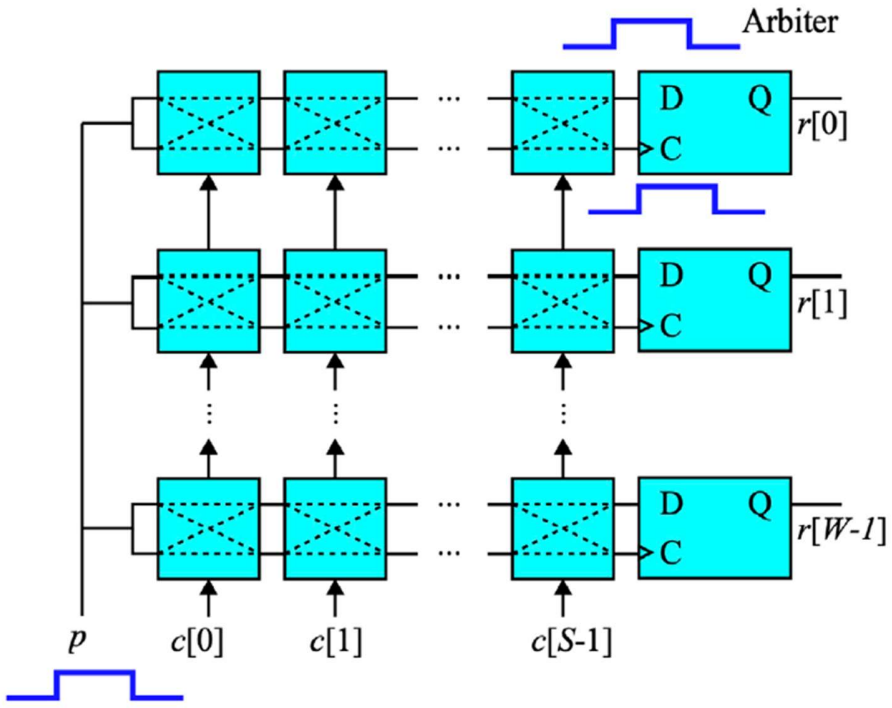


Figure 2.1 | Structure for multiple row arbiter PUF consisting of W arbiter rows and S 2 × 2-switching stages per row [7]

One enable signal is attached to the first switch block's input [7]. There are three inputs and two outputs for every switch block. As seen in Figure 2.2, a switch block comprises of two 2-to-1 multiplexers (MUXs) that share the same selection bit c[i]. The routes will cross if the challenge bit is "1", the trajectories will be parallel if not. An arbiter is attached to the last switch block's two outputs. The arbiter may be a S R latch or a flip-flop that produces a one-bit response as a result of the variations in the times at which the outputs from the last n switch blocks arrive.

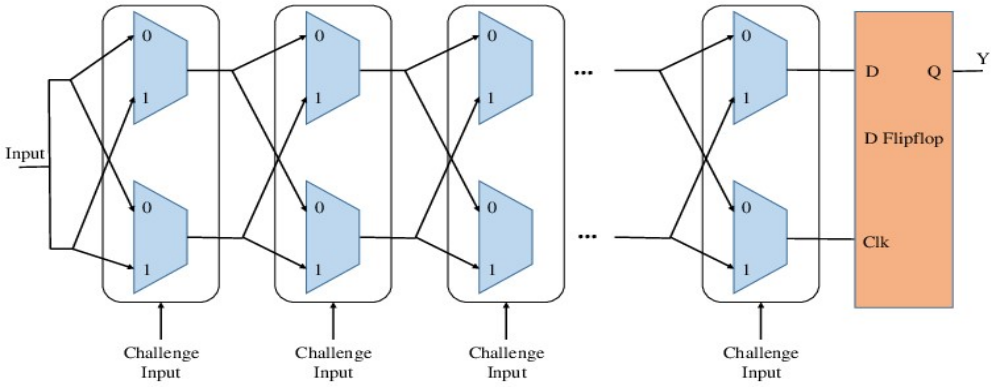


Figure 2.2 | Switch block structure of single row with Arbiter Operation of D flipflop [10]

Figure 2.2, shows the Switch block structure of single row with Aribter operation of D flip-flop. The response will be "1" if the top data input D arrives first. The response will be "0" if the lower clock signal arrives first.

2.3. Applications of PUF

This section illustrates the device authentication, cryptography, and intellectual property storage capabilities of the PUF framework.

2.3.1 Cryptography

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. [11] These secret keys are stored in a non-volatile memory that attackers can easily access. The hardware security primitive, Physical Unclonable Function (PUF) is a promising alternative for enhancing the security of interconnected devices. A PUF produces an output in response to an input based on the physical structure and intrinsic manufacturing variations of an integrated circuit (IC). The generated random response being unpredictable, act as a robust secret key in cryptographic protocols.

2.3.2. Intellectual Property Storage

Another crucial use of PUF is for the protection of intellectual property (IP) [12]. The intellectual property is encrypted using a private key generated using the PUF architecture. The design is decrypted using a public key, which is also produced by the PUF design. An opponent attempting to reverse engineer the chip's architecture won't be able to generate the same Cryptographic keys because PUF design generates private and public keys at random. The attacker is thus prevented from releasing an unauthenticated gadget onto the market.

2.4. Threats on PUF-Based IoT Devices

This section examines potential dangers to IoT devices that rely on PUFs for authentication. IoT devices may be set up in unsecured locations. They are consequently exposed to a range of dangers, such as direct physical assaults (such as hammering), communication disruptions, and manipulation of the actual operating circumstances of the device. Additionally, more conventional attacks like attempting to read secret keys from memory and communication flaws are feasible.

The key new threat to a PUF-based security system is for an attacker to acquire the capacity to offer the appropriate response to a specific challenge. This can be accomplished, for instance, by building a physical replica of the PUF or by using a modelling attack to predict CRPs. We go into further depth on both strategies later. We take into account two various attacker models. In the first, the attacker has the ability to eavesdrop on device communication. In the second, the device is physically accessible to an enemy.

2.4.1. Man in the Middle Attack

An attacker can listen in on the communication channel between a server and a device in this kind of attack, intercept, and store the CRPs that are being sent. Following that, CRPs can be exploited directly to carry out replay attacks or indirectly by feeding them into a machine learning algorithm and creating a model of the PUF that can anticipate other CRPs.

Since devices frequently link dynamically to previously unidentified devices in the IoT, man in the middle attacks can be carried out with comparably little effort. An adversary can put a cheap computer, such as a Raspberry Pi, close to the device being attacked and allow it to connect to the same (potentially encrypted) wireless network. An attacker can listen in on the communication channel between a server and a device in this kind of attack, intercept, and store the CRPs that are being sent. Following that, CRPs can be exploited directly to carry out replay attacks or indirectly by feeding them into a machine learning algorithm and creating a model of the PUF that can anticipate other CRPs.

Since devices frequently link dynamically to previously unidentified devices in the IoT, man in the middle attacks can be carried out with comparably little effort. An adversary can put a cheap computer, such as a Raspberry Pi, close to the device being attacked and allow it to connect to the same (potentially encrypted) wireless network. Therefore, the risk for such an attack taking place is relatively high, and suitable defence mechanisms must be used.

2.4.2 Side Channel Attack

The attacker in this kind of assault has direct physical access to the target device. Two orthogonal axes can be used to classify side channel assaults on PUF. We can distinguish between invasive, semi-invasive, and non-invasive attacks on the first axis. Attacks can be active or passive on the second axis.

2.4.2.1 Invasive, Semi-invasive, and Non-Invasive Attacks

By destroying the chips and having direct access to the internal components, invasive attacks can be carried out. PUFs were formerly thought to be impervious to such attacks because they would weaken a PUF's structure and render it non-operational. No on-chip private key storage exists that an attacker may access. PUFs are in reality susceptible to invasive attacks, such as the production of a complete physical clone of a PUF, according to reports. Invasive attacks are significantly more expensive and complicated. The IoT device being attacked typically needs to be moved into a specialised lab with pricey lab equipment. Due to this, IoT devices are less appealing targets for this type of assault, especially when they are placed in public areas and transmit sensitive information.
An attacker needs access to the chip surface for semi-invasive attacks, but the passivation layer of the semiconductor is unaffected. Semi-invasive PUF attacks have been described using electromagnetic probing and photonic emission. Semi-invasive attacks use less complex methods than invasive attacks, although they still need specialised lab equipment.
Non-invasive attacks aim to obtain sensitive data by taking advantage of data (such as power usage or delay time) without having direct access to internal components. The necessary equipment for these attacks is reasonably portable and inexpensive, and it can even be put close to the IoT devices being attacked. Machine learning algorithms are used by non-invasive attacks as an analysis tool and can accurately recreate CRPs.

2.4.2.2. Active and Passive Attacks

Active attacks aggressively alter the system, such as by changing the operational temperature or the supply voltage V_{cc} to carry out attacks on the PUF. In contrast, passive attacks target a PUF by passively observing data, such as the PUF's temperature or energy usage. Both forms of attacks have been demonstrated to be relevant to PUF attacks and both involve direct physical access to the target device.

2.5 Defence Strategy

We described attacks on PUF-based IoT devices in the preceding section. Every day a fresh approach is put out. We now go over potential defence tactics for the various attack types. Attacks using side channels come first. Attacks through side channels that are invasive or semi-invasive demand direct access to the PUF. If the enemy is able to transport the device to a lab, (s)he can employ cutting-edge equipment to assault the system. The threat level can be reduced but not completely eliminated by adequate physical protection mechanisms for IoT devices (such as epoxy adhesive or attaching the PCB) and anomaly detection (such as by identifying anomalous device motion using pricey gyroscope sensors).

However, keep in mind that these assaults are typically expensive, and cost effectiveness can be seen as a hurdle for inexpensive IoT devices. On the other hand, non-invasive attacks can be effective outside of a lab and without harming the device's physical security with relatively inexpensive equipment. In order to build a machine learning-based model of the PUF, passive non-invasive assaults apply a number of challenges to the PUF and keep track of external elements like power consumption. To accomplish this, thousands or maybe hundreds of thousands of hurdles must be overcome.

To combat this, the PUF can be designed to either only take very particular challenges, making it exceedingly challenging to acquire enough of them, or to only accept a tiny number of challenges every second, greatly slowing the attack. Additionally, active non-invasive attacks take advantage of PUFs' potential for varying behaviour depending on the operational environment. They can, for instance, change these conditions to decrease the number of useful CRPs and to simplify modelling attacks. Making the PUF more resistant to outside influences would effectively put active attacks in the same category as passive attacks while still providing defence against the modelling attack itself.
Avoiding the reuse of is a well-known defence against replay attacks for man in the middle attacks. There are two possibilities First, CRPs can be encrypted to maintain their secrecy. This is especially useful when there are few CRPs available, such as for weak PUFs, but it uses additional computational resources, negating our original motivation for using PUFs in the IoT. As an alternative, we can select a (strong) PUF design with a sufficient number of CRPs to ensure that none are ever reused. The traditional method for increasing the number of CRPs is to use more of the chip's computational resources. This method might not work well because IoT devices have limited processing resources.
In order to increase the number of CRPs without expanding the size of the PUF circuit on the chip, a method utilising a spatial reconfigurable PUF has recently been proposed. Keep in mind that for added security, we can combine powerful PUFs with encrypting CRPs. But this will result in more overhead once more.

2.6 Evaluation Metrics for PUF

The performance of PUFs can be evaluated with three parameters: uniformity, uniqueness, and reliability [13] [14][15].

2.6.1 Uniformity

The degree to which the proportion of "0" and "1" in a PUF's replies is uniform is referred to as uniformity [15]. The probability of "1"s should ideally equal the probability of "0"s for PUF responses to be unpredictable and random. The fractional Hamming Weight (HW) of the total replies is used to calculate uniformity as given by [16]:

$$Uniformity = \frac{1}{m} \times HW(R) \times 100\% \quad (1)$$

Where Ri is the m-bit responses on chip i and HW(Ri) is the number of ‘1’s in the responses. The value is supposed to be close to 50%.

2.6.2 Uniqueness

The ease with which one PUF instance can be identified from another PUF instance is characterised by uniqueness [15]. Since inter-chip differences are measured by uniqueness, every pair of chips should be taken into account. The average fractional inter-chip Hamming Distance (HD) between responses produced in response to the identical challenges from several chips can be used to evaluate it as given by [16]:

(2)

Where k is the number of chips, Ri and Rj represent-bit responses generated on different chips i and j ($i \neq j$) respectively. For the uniqueness of the PUF on each device, the value should approach 50% [15].

2.6.3 Reliability

The ability of a PUF to replicate the responses during various measurements is characterised by reliability [15]. Multiple times, the same amount of answers is extracted in different environmental conditions, such as temperature and supply voltage.
Reliability is estimated using the average fractional intra-chip HD of replies produced by the same challenges on the same chip as given by [16]:

$$HD_{INTRA} = \frac{1}{k}\sum_{i=1}^{k1}\frac{HD(R_i(n), R_I'(n))}{n} \times 100\% \quad (3)$$

Where x is the number of samples to be collected on chip i in the same environment, Ri is the reference m-bit responses of chip i in some environmental condition, and R0i;y is the responses of the k sample in a different condition.
The reliability of the PUF on chip *i* is defined below as given by [16]:

$$reliability = 100\% - HD_{INTRA} \quad (4)$$

Ideally, HD_{intra} is expected to be 0 while the reliability is 100%, which means the PUF is totally reliable.

$$Uniqueness = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\frac{HD(R_i, R_j)}{n} \times 100\%$$

CHAPTER 3

PUF BASED RANDOM NUMBER GENERATION

In a lot of computer science solutions, random numbers are crucial components. To ensure limitations on computational complexity, randomised algorithms need a random source, and sampling techniques frequently depend on randomization to appropriately reflect the data they are collecting [16].

Random numbers are used in cryptographic applications to produce encryption keys, establish starting parameter values, and include random nonces into protocols and padding methods. These random numbers are typically generated by a pseudorandom number generator (PRNG), a deterministic software algorithm that mimics randomness.

3.1 Randomness

An unbiased "fair" coin with sides labelled "0" and "1" and a probability of exactly 1/2 yielding a "0" or "1" on each flip might be used to simulate a random bit sequence. The coin flips are also independent of one another; the outcome of one flip does not influence another. Given that the distribution of "0" and "1" values will be determined by chance, the impartial "fair" coin is the ideal random bit stream generator (and [0,1] uniformly distributed). No matter how many elements have already been produced, since each element in the sequence is formed independently of the others, it is impossible to predict what value will be contained in the following element.

3.2 Unpredictability

For cryptography applications, generated random and pseudorandom numbers should be unpredictable. In the case of PRNGs, the next output number in the sequence should be unpredictable if the seed is unknown, regardless of any prior knowledge of the sequence's random numbers. Forward unpredictability is the name for this characteristic. Additionally, knowing the seed from knowledge of any generated values should not be possible (i.e., backward unpredictability is also required). Each element of the sequence should appear to be the result of an independent random occurrence with probability 1/2; there should be no correlation between a seed and any value derived from that seed.

3.3 Random Number Generators (RNGs)

A random number generator is the earliest kind of sequence generator (RNG). An RNG generates randomness by using an entropy source, a nondeterministic source, and a processing function (the entropy distillation process). Any entropy source flaw that causes the generation of non-random numbers must be remedied by the employment of a distillation procedure (e.g., the occurrence of long strings of zeros or ones). A physical quantity serves as the typical entropy source, such as the noise in an electrical circuit, the timing of user actions (such as key presses or mouse movements), or the quantum effects in a semiconductor. It is possible to use different combinations of these inputs.

The results of a RNG can be fed into a pseudorandom number generator or utilised directly as random numbers (PRNG). In order to establish that the physical sources of the RNG inputs appear random, any RNG's output must pass stringent randomness criteria as determined by statistical tests in order to be employed directly (i.e., without further processing). For instance, a physical source like electronic noise may include a superposition of regular patterns like waves or other periodic events. These structures may appear to be random, but statistical analyses show that they are not random.

3.4 Pseudorandom Number Generators (PRNGs)

A pseudorandom number generator is the second type of generator (PRNG). A PRNG creates several "pseudorandom" numbers using one or more inputs. PRNG inputs are referred to as seeds. When unpredictable behaviour is required, the seed itself must be unpredictable and random. So, by default, a PRNG should get its seeds from a RNG's outputs; in other words, a PRNG needs a RNG to work with it.
A PRNG's outputs are often deterministic functions of the seed, meaning that the only genuine unpredictability occurs during seed creation.
The name "pseudorandom" comes from the process' deterministic character. Each component of a pseudorandom sequence can be recreated from its seed, hence only the seed needs to be saved if the pseudorandom sequence is to be reproduced or validated.

Contrary to popular belief, pseudorandom numbers frequently seem to be more random than random numbers generated physically. When a pseudorandom sequence is built correctly, each value is created from the one before it through changes that seem to add more randomness. These statistical auto-correlations between input and output can be removed through a series of such changes. As a result, a PRNG may produce outputs faster than a RNG and with superior statistical features.

3.5 Shannon Entropy

The entropy of an object or a system is a measure of the randomness within the system.
Shannon entropy[17], commonly referred to as information entropy or the Shannon entropy index, is a measure of how random a set of data is. It is used to gauge the degree of uncertainty surrounding the placement of a particular character in a string of text after another. The more characters there are or the more equal the rates of occurrence, the harder it would be to predict what would happen next, raising the entropy. When an outcome is known for sure, entropy is zero.

However, a key aspect of the Shannon entropy equation; that multiplying each possible sequence's code length by its probability gives the code length we would need to send on average.
Given a discrete random variable X, with possible outcomes x1,...,xn, which occur with probability P(x1),...,P(xn), the entropy of X is formally defined as given by [17]:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i) \tag{5}$$

Where sigma denotes the sum over the variable's possible values. Usually, the log here is base 2, but changing the logarithm will change what unit of entropy you get. Therefore, in order to calculate Shannon Entropy you need to first get all of the probability of your random variable X.

3.6 Linear Feedback Shift Registers

The pseudo-random bit sequences can be generated using LFSRs[18]. A common internal structure of LFSR consists of: D flip-flops that form a simple shift register and a number of feedback loops collected into XOR gate. Advantages of LFSRs include the ease of implementation, simplicity, speed, and the ability to generate a maximal cycle sequence with the same uniform statistical distribution of 0's and 1's as in a truly random sequence.

3.7 Von Neumann corrector

Von Neumann correction to extract randomness from the temporal ordering of ones and zeros. Von Neumann Correction is a process for removing bias from pseudo-random bit streams[19]. Von Neumann correction takes pairs of bits as inputs but output bits only when there is a transition in the input bit stream. Because transitions in each direction occur an equal number of times, the bias is eliminated. This comes at the cost of a lower bit rate.
The algorithm works on pairs of bits, and produces output as follows:

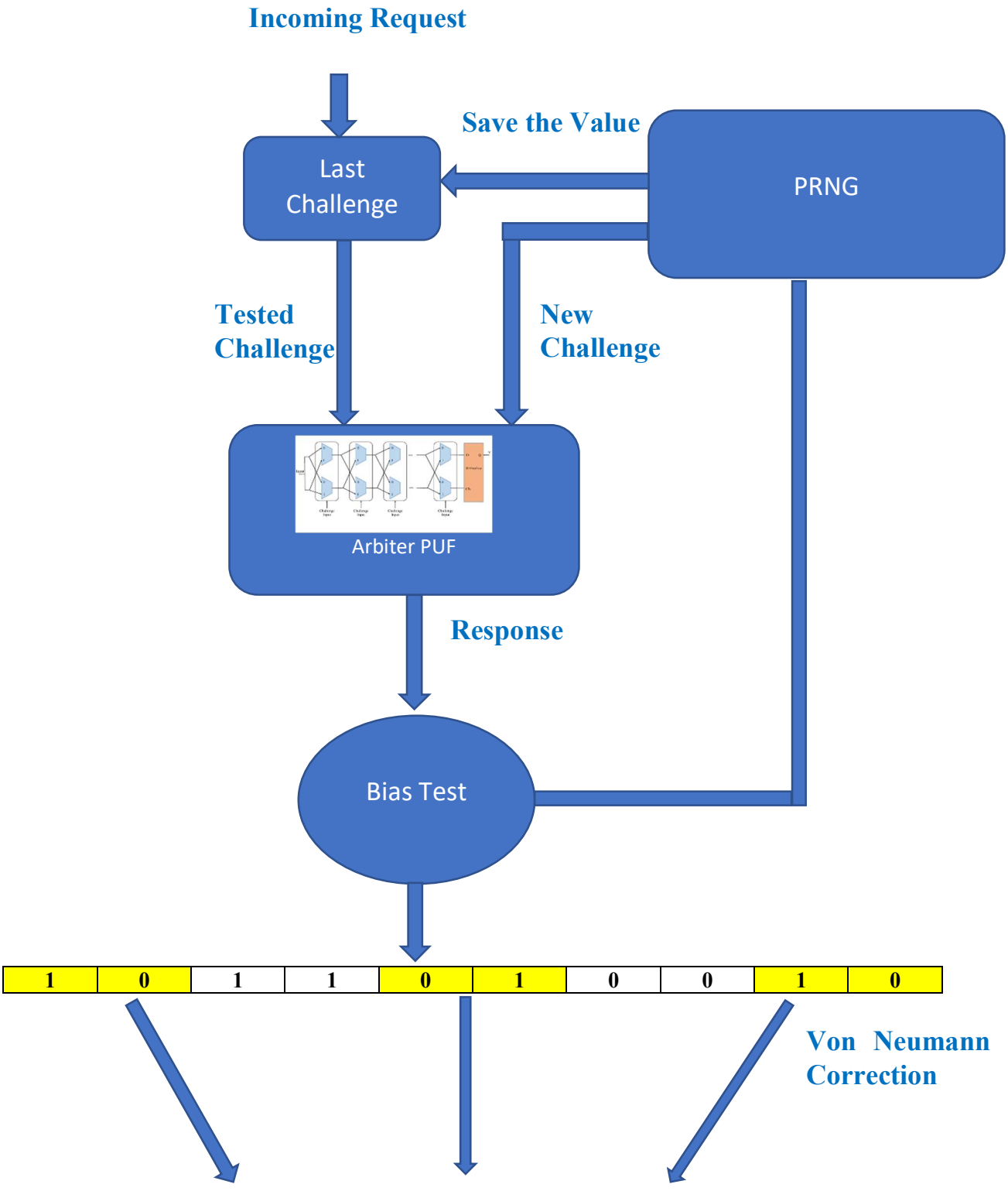
If the input is "00" or "11", the input is discarded (no output).
If the input is "10", output a "1".
If the input is "01", output a "0".
In this way we parse the N bits until 32 random bits have been determined.

3.8 Random Number Generation

System accepts a single input request for a random output and produces an output using an iterative method that identifies a challenge which produces unpredictable results that means when this challenge is applied to PUF it will produce ones with probability of $0.5\pm\epsilon$ [20].
The challenge is fed to the arbiter PUF (APUF), which is shown in figure 3.1. It is a delay-based PUF since the response is generated based on the intrinsic timing differences of two topologically and functionally identical paths in an IC due to its manufacturing variations. Two electrical pulses race simultaneously through two paths consisting of several stages. Each stage consists of a pair of multiplexers, which are determined by a challenge Ci, configured as a crossbar switch. A latch or flip-flop acts as an arbiter to determine which of the signals along the two symmetrical paths is faster. When the output signal in the upper path is faster, the arbiter outputs 1; otherwise, it outputs 0. Therefore, an APUF can convert the delay differences into digital responses controlled by the challenges. For an N-stage arbiter PUF, it can generate 2N Challenge-Response Pairs (CRPs).

The output response bits of an APUF are examined to check if they seem to create ones with a probability of $0.5\pm\epsilon$ that means the probability of output response to be one should be $0.5\pm\epsilon$ The challenge feed to the PUF N times to create N bits response and verify that $N/2\pm\varepsilon$ of the responses were ones, while the rest were zeros. If the challenge fails this bias test, challenge is fed to pseudorandom number generator to create a

new challenge to be tested. If the challenge passes this bias test, then this challenge is saved as an unpredictable challenge to be used. A straightforward PRNG can be built using a linear feedback shift register with that saves the subsequent value. Finally, 32 random bits are derived from these N bits by applying a Von Neumann corrector.



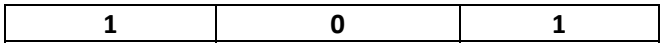


Figure 3.1 | PUF Random Number Generation [20]

CHAPTER 4

ANALYSIS OF ARBITER PHYSICAL UNCLONABLE FUNCTION

This chapter provides the analysis of Aribter PUF. Authentication algorithms including the proposed algorithm and the result histogram of each algorithm. A PUF is primarily used for device authentication and for securely generating secret keys. A PUF can be conceptualised as a circuit element that produces a response r in response to the application of a challenge c:

$$r = F(c)$$

4.1 Statistical Analysis

The random variable chosen for modelling arbiter PUF is the single switching stage delay τ in the upper or lower outputs [7]. We can write τ as the sum of two random processes given by [7]:

$$\tau = \tau_p + \tau_n \tag{6}$$

where τ_p is due to static random process variation (RPV) and τ_n is due to dynamic CMOS noise.

The variable τ_p follows the biased Gaussian process whose pdf is given by [7]:

$$f_{T_p}(\tau_p) = \frac{1}{\sigma_p\sqrt{2\pi}} e^{-(\tau_p-\mu_p)^2/2\sigma_p^2} \tag{7}$$

where μ_p is the mean and σ^2_p is the variance of the RPV process.

The variable τ_n follows a zero-mean Gaussian process in equation (8) given by [7]:

$$f_{T_n}(\tau_n) = \frac{1}{\sigma_n\sqrt{2\pi}} e^{-\tau_n^2/2\sigma_n^2} \tag{8}$$

where σ^2_n is the variance of the dynamic random CMOS noise process.

The combined effects of RPV and CMOS noise for a specific PUF generate a pdf given by [7]:

$$f_T(\tau) = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-(\tau-\tau_p)^2/2\sigma_n^2} \tag{9}$$

where τ_p is the contribution of RPV and σ_n is the contribution of dynamic random CMOS noise.

The total delay of a ring-oscillator is the sum of S switching stages is given by [7]:

$$T = \sum_{s=1}^{S} \tau(s) = \sum_{s=1}^{S} \left[\tau_p(s) + \tau_n(s) \right] \tag{10}$$

the first moment, or expected value, for the delay of the arbiter PUF is given by [7]:

$$\begin{aligned} <T> &= <\sum_{s=0}^{S-1} \tau(s)> \\ &= \sum_{s=0}^{S-1} <\tau(s)> \\ &= S\,\mu_p \end{aligned} \tag{11}$$

The variance of the total delay can be found as given by [7]:

$$\begin{aligned} \sigma^2 &= <(T - S\mu_p)^2> \\ &= S\left(\sigma_p^2 + \sigma_n^2\right) \end{aligned} \tag{12}$$

4.2 Encoding of the Arbiter Physical Unclonable Functions Response

Generating a digital signature from the response of the Arbiter PUF is obtained directly by choosing B arbiters. The arbiter outputs r(j) are measured with $0 \leq j < B$. The values obtained represent the desired response.

4.3 Simulation of Arbiter PUF

In this section, we discussed the simulation of arbiter PUF. Millions of devices produced at fabrication house. In order to simulate these devices, fabricator selects a specific challenge c. This specific challenge is fed to the devices and response bits r generated. Thus, the combination of a challenge and its corresponding response a Challenge Response Pair (CRP) is generated. All devices generates unidentical responses for a particular challenge.

Funtion for generating response of device is given below:

A typical system will have W rows and each row has S 2 × 2-switching stages and an SR latch or D-type flip-flop (D-FF) acting as the arbiter. When bit c(j) 0, the switching stage is in the straight through setting where the signal at the upper input is routed to the upper output. The signal at the lower input is routed to the lower output. When bit c(j) 1, the switching stage is in the cross setting where the signal at the upper input is routed to the lower output. The signal at the lower input is routed to the upper output.

The procedure for obtaining CRP data proceeds as follows:

1. The challenge word c is chosen which consists of S bits to configure the state of the switching stages.

2. A pulse p is issued to all the arbiter rows.

3. B bits are selected to check the content of the output arbiters and this is considered the response r.

Table 4.1 illustrates the response generation function which calculates the device response using upper and lower path delays.

Require: t_setup, Total_delay_upper, Total_delay_lower
1: Delay_difference= Total_delay_upper-Total_delay_lower *% Delay difference of upper and lower path*
2: **if** | Delay_difference | > t_setup *% If the setup-hold time tsetup is smaller than the time difference between the upper and lower output pulses from the switching stage at position S-1.*
3: **if** Delay_difference > 0 *% If the time difference between the upper and lower output pulses from the switching stage at position S-1 is greater than 0.*
4: response=0 *% Output response will be 0.*
5: **else** *%Else if the time difference between the upper and lower output pulses from the switching stage at position S-1 is lesser than 0.*
6: response=1 *% Output response will be 1.*
7: **end if**

8: **end if**
9: **return** response *% Response will be generated.*

Table 4.1 | Pseudocode for response generation function.

L1: The delay difference between upper and lower path calculated.

L2: Checks if the setup-hold time tsetup is smaller than the time difference between the upper and lower output pulses from the switching stage at position S-1.

L3: Checks if the time difference between the upper and lower output pulses from the switching stage at position S-1 is greater than 0.

L4: If the time difference is greater than 0 then the output response is 0.

L5: But if the time difference between the upper and lower output pulses from the switching stage at position S-1 is lesser than 0.

L6: If the time difference is lesser than 0 then the output response is 1.

L9: Output response is generated.

Function for Generating golden response of device is given below:

The golden response for a physical unclonable function (PUF) is obtained during the manufacturing or testing process of the PUF. During the manufacturing or testing process, the PUF is characterized by applying a large number of random challenges to the PUF and recording the corresponding responses. These challenge-response pairs (CRPs) are used to create a database of expected responses for the PUF. The golden response is obtained by applying a pre-selected challenge to the PUF and computing the response based on the model that was generated during the characterization process. This response is then stored as the expected or correct response for that particular challenge. In order to obtain an accurate and reliable golden response, multiple iterations or trials may be required to account for variations in the physical characteristics of the PUF and to ensure that the response generated is consistent and reliable. During each iteration, a different random challenge is presented to the PUF, and the corresponding response is recorded. The number of iterations required to obtain a reliable and accurate golden response for a PUF should be determined through careful testing and analysis. Different PUF designs and applications may require different numbers of iterations to achieve the desired level of security and reliability.

It is necessary to receive the specific device characteristics from the manufacturer and then share these qualities with a reliable certification authority in order to use the PUF for authentication and secure key exchange (CA). The steps that the maker must take to get each device's "golden response" are as follows:

1. A set of challenge words C and a number of iterations N are defined.

2. A chosen challenge word $c \in C$ is applied N times to the PUF in the device.

3. At iteration n, the response r(n) defining the PUF output is measured.

4. The average or golden mean value $\mu_g = < r(n) >$ and variance

$\sigma^2_g = < (r(n) - \mu_g)^2 >$ are obtained.

5. The encoding scheme associated with the PUF is used to obtain the golden response r_g = PUF_Encode (μ_g). For example the encoding scheme for the Arbiter PUF is given by [7]:

$$r_g = \begin{cases} 0 & \text{when} \quad \mu_g < 0.5 \\ 1 & \text{when} \quad \mu_g \geq 0.5 \end{cases} \tag{13}$$

6. The golden variance value σ2 g is used as a guideline to estimate the number of bits in error for a given B-bit response word r. This estimate is used to obtain the redundant data w = FEC_Encode (r_g, σ_g). Where FEC_Encode is a forward error correcting block coding scheme.

Table 4.2 illustrates that the Golden responses are generated using the upper and lower limit of setup and hold time. Upper limit and lower limit of setup and hold time are assumed to be 0.7 and 0.3 respectively. If the response is above the upper limit of t setup and hold time (0.7 considered) then the valid and fake golden response will be considered as l and if the response is below the lower limit of t setup and hold time (0.3 considered) then the valid and fake golden responses will be considered as 0.

If the response is in the range of setup and hold time 0.3 to 0.7 then the PUF is in metastable stage. In metastability condition, flag response will be 1. In this metastability condition, golden response shall be randomly 1 or 0.

```
Require: response, N % Number of Iterations
Generate g_response= average of N iterations  (response)      % Average of responses of all N iterations.
    if (0.7<=g_response || g_response <=0.3)       % Average response range 0 to 0.3 and 0.7 to 1.
        if g_response >=0.7      % if average response is greater than or equal to 0.7.
        Golden_response = 1      % Golden response will be 1.
        else if g_response <= 0.3   % if average response is lesser than equal to 0.3.
        Golden_response = 0      % Golden response will be 0.
        end if
    else      % if g_response is in range of 0.3 to 0.7.
        g_response= probability
          probability=random          % response will be based on random probability.
            if probability<0.5         % if random probability is less than 0.5.
               Golden_response =1      % Golden response will be 1.
            else
               Golden_response =0      % Golden response will be 0.
            end if
end if
Return Golden_response          % Golden respone will be generared.
```

Table 4.2 | Pseudocode for Golden response generation function.

L1: The response average is calculated for N iterations.

L2: Check if average response range 0 to 0.3 and 0.7 to 1.

L3: Check if average response is greater than or equal to 0.7.

L4: If average response is greater than or equal to 0.7 then golden response is 1.

L5: Check if average response is lesser than or equal to 0.3.

L6: If average response is lesser than or equal to 0.3 then golden response is 0.

L8: But if average response is in range of 0.3 to 0.7.

L9: Golden response is a random probability between 0 to 1.

L11: If random probability is less than 0.5.

L12: Golden response is 1.

L13: If random probability is greater than 0.5.

L14: Golden response is 0.

L17: Golden response is generated.

Function for Generating golden response of device is given below:

Table 4.3 illustrates the Hamming distance calculation between the device responses. Hamming distance is defined as the sum of absolute differences between the device responses.

Function for Hamming Distance

Require: responses
1: abs_difference= |difference in responses| *% difference in responses of two devices.*
2: Hamming_distance=sum(abs_difference) *% sum of absolute differences of responses.*
3: return Hamming_distance *% Hamming distance is calculated.*

Table 4.3 | Pseudocode for Hamming distance calculation function.

L1: Absolute difference between two device responses are calculated.

L2: Sum of absolute difference is calculated.

L3: Hamming distance is calculated.

4.3.1 Algorithm-1

In this algorithm, the single-challenge approach used to authenticate a device using the following basic technique. This algorithm is designed on basic Four steps are required for authenticating the device and generating the session key.

#1: Fabricator selects a single CRP (c, w)

#2: Fabricator generates r, K and h

#3: Field devices uses (c, r) to generate w', K, and h*

#4: Fabricator authenticates device

Algorithm #1 is vulnerable to effects of thermal noise which leads to large intra Hamming distance and small inter Hamming distance.

The purpose of this algorithm is to perform a simulation. One device is selected for simulation at fabrication house called a Fabrication device and the same device at field called a valid device. One other device is called a fake device.

This Algorithm-1 [7] requires the Fabrication response, Valid response and Fake response. Intra Hamming distance between fabrication device response and valid device response and Inter Hamming distance between fabrication device response and fake device response is calculated.

The distance between intra and inter hamming distance is measured between maximum value of intra hamming distance and minimum value of inter hamming distance.

Table 4.4 illustrates the Algorithm-1, Generation of responses for all devices using response function and Calculation of Hamming distance using Hamming distance function.

Require: t_setup, Fab_delay_upper, Fab_delay_lower, Valid_delay_upper, Valid_delay_lower, Fake_delay_upper, Fake_delay_lower

1: Fab_response= **response** (t_setup, Fab_delay_upper, Fab_delay_lower) *%Response generated for selected device at Fabrication house.*

2: Valid_response= **response** (t_setup, Valid_delay_upper, Valid_delay_lower) *%Response generated for selected device at field.*

3: Fake_response= **response** (t_setup, Fake_delay_upper, Fake_delay_lower) *%Response generated for other device at field.*

4: Intra1_HD = **Hamming_distance** (Fab_response, Valid_response) *%Hamming distance calculated between selected device at Fabrication house and same selected device at Field.*

5: Inter1_HD = **Hamming_distance** (Fab_response, Fake_response) *%Hamming distance calculated between selected device at Fabrication house and other device at Field.*
6: **Return** diff= Min. of Inter1_HD – Max. of Intra1_HD

Table 4.4 | Pseudocode for Algorithm-1.

L1: Response generated for selected device at Fabrication house.

L2: Response generated for selected device at field.

L3: Response generated for other device at field.

L4: Intra Hamming distance calculated between selected device at Fabrication house and same selected device at Field.

L5: Inter Hamming distance calculated between selected device at Fabrication house and other device at Field.

L6: Difference of Intra and Inter Hamming distance calculated.

Require: Intra1_HD , Inter1_HD
1: Histogram for Intra Hamming distance= **Histogram** (Intra1_HD)*%Hamming distance calculated between selected device at Fabrication house and same selected device at Field.*
2: Histogram for Inter Hamming distance= **Histogram** (Inter1_HD)) *%Hamming distance calculated between selected device at Fabrication house and other device at Field.*

Table 4.5 | Pseudocode for Plotting Histogram.

L1: Histogram plotted for the Intra Hamming distance calculated between selected device at Fabrication house and same selected device at Field.

L2: Histogram plotted for the Inter Hamming distance calculated between selected device at Fabrication house and other device at Field.

Algorithm-1 with 8-bits response

Algorithm-1 is evaluated by taking 8-bits response and histogram is plotted as shown in figure 4.1. This figure shows that intra hamming distance and inter hamming distance bar overlap each other in 8 bits response.

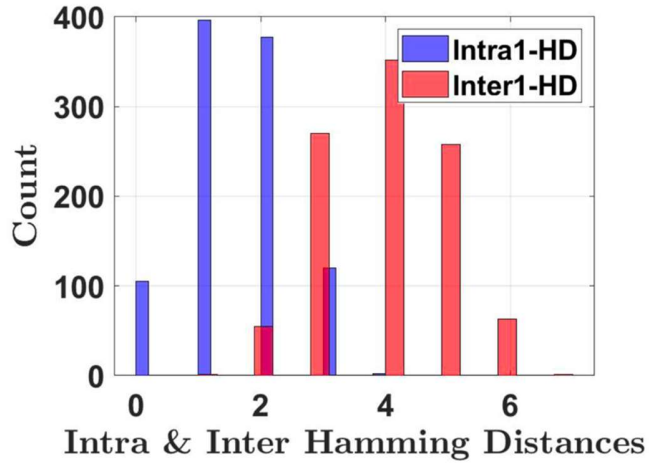


Figure 4.1 | Algorithm 1- Inter & Intra Hamming Distances for B = 8-bits.

Algorithm-1 with 16-bits response

Algorithm-1 is evaluated by taking 16-bits response and histogram is plotted as shown in figure 4.2. This figure shows that intra hamming distance and inter hamming distance bar overlap each other in 16 bits response.

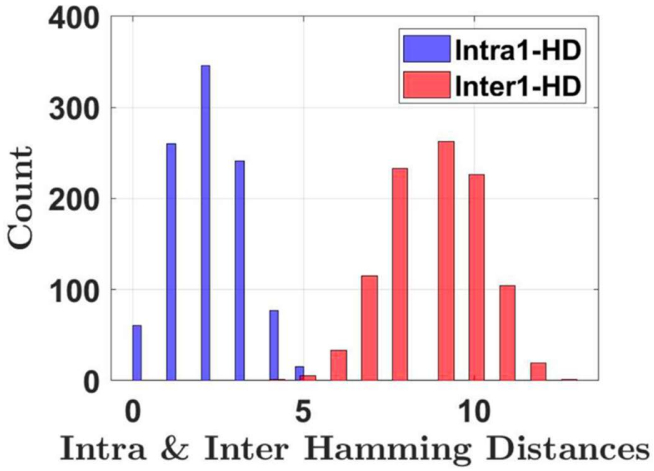


Figure 4.2 | Algorithm 1- Inter & Intra Hamming Distances for *B* = 16-bits.

Algorithm-1 with 32-bits response

Algorithm-1 is evaluated by taking 32-bits response and histogram is plotted as shown in figure 4.3. This figure shows that intra hamming distance and inter hamming distance bars overlap each other in 32 bits response.

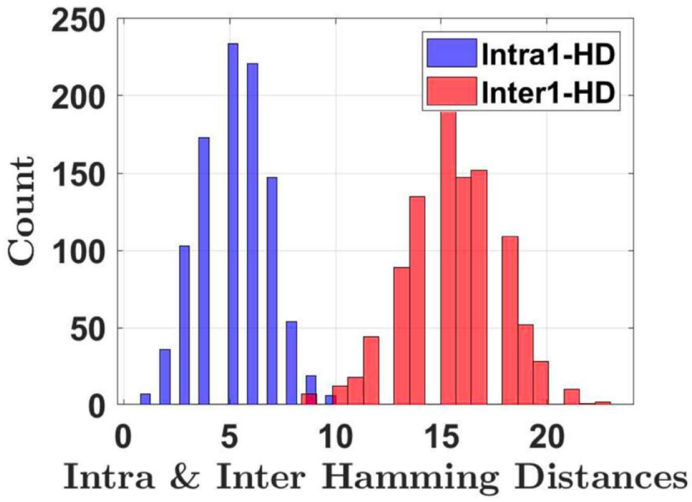


Figure 4.3 | Algorithm 1- Inter & Intra Hamming Distances for B = 32-bits.

Algorithm-1 with 64-bits response

Algorithm-1 is evaluated by taking 64-bits response and histogram is plotted as shown in figure 4.4. This figure shows a 2 bits difference between intra hamming distance and inter hamming distance in 64 bits response.

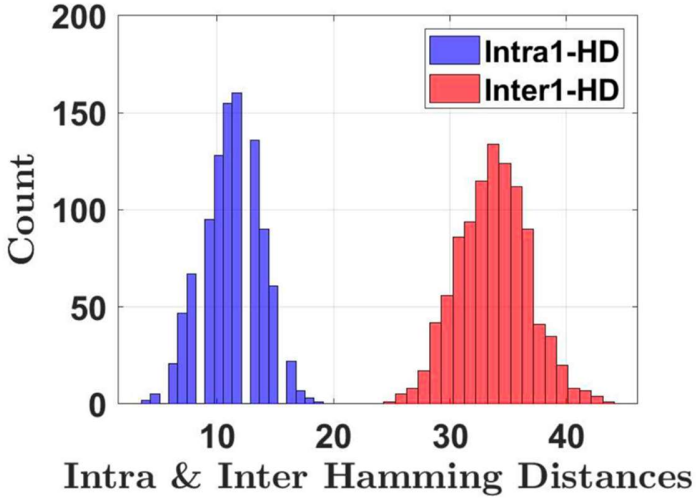


Figure 4.4 | Algorithm 1- Inter & Intra Hamming Distances for B = 64-bits.

Algorithm-1 with 128-bits response

Algorithm-1 is evaluated by taking 128-bits response and histogram is plotted as shown in figure 4.5. This figure shows approximately 12 bits difference between intra hamming distance and inter hamming distance in 128 bits response. Therefore, in 128 bit reponse valid and fake device can be identified.

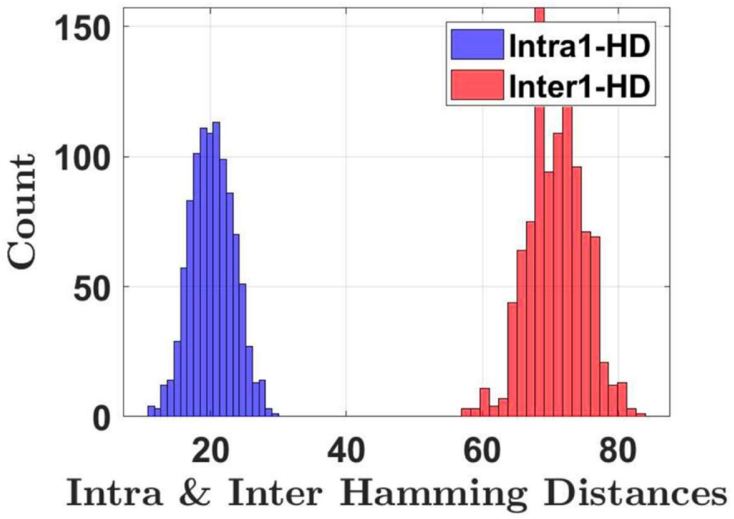


Figure 4.5 | Algorithm 1- Inter & Intra Hamming Distances for B = 128 -bits.

4.3.2 Algorithm-2

The purpose of Algorithm-2 [7] is to eliminate thermal noise by repeating the steps used by the Fabricator to obtain the golden response. Algorithm-2 perform N initializations of the Arbiter PUF and prepares an N x B matrix W' whose rows are the individual responses w'[n] for the challenge c. Four steps are required for authenticating the device and generating the golden response.
#1: Fabricator selects a CRP (c, w, N)
#2: Fabricator generates r, K and h
#3: Field devices uses (c, r, N) to generate w', K, and h*
#4: Server authenticates device

One device is selected for simulation at fabrication house called a Fabrication device to obtain the golden response. Same challenge is applied to the selected device N times to get the N responses. The same device at field is called a valid device. One other device is called a fake device.

The distance between intra and inter hamming distance is measured between maximum value of intra hamming distance and minimum value of inter hamming distance.

Table 4.6 illustrates the Algorithm-2 in which response generation will be same as algorithm-1. Golden Response is generated at Fabrication House using golden response function and Hamming distance is calculated using Hamming distance function.

Require: Fab_response

1: Fab_Golden_response= **Golden_response**(Fab_ response)
2: **Return** Fab_Golden_response
3: Intra2_HD = **Hamming_distance** (Fab_Golden_response, Valid_response)
4: Inter2_HD = **Hamming_distance** (Fab_Golden_response, Fake_response)
5: **Return** difference= Min. of Inter2_HD – Max. of Intra2_HD

Table 4.6 | Pseudocode for Algorithm-2.

L1: Golden Response generated for selected device at Fabrication house.

L3: Intra Hamming distance calculated between Fabrication golden response and same selected device at Field.

L4: Inter Hamming distance calculated between Fabrication golden response and other device at Field.

L5: Difference of Minimum Inter and Maximum Intra Hamming distance calculated.

Require: Intra2_HD , Inter2_HD 1: Histogram for Intra Hamming distance= **Histogram** (Intra2_HD)*%Hamming distance calculated between selected device at Fabrication house and same selected device at Field.* 2: Histogram for Inter Hamming distance= **Histogram** (Inter2_HD)) *%Hamming distance calculated between selected device at Fabrication house and other device at Field.*

Table 4.7 | Pseudocode for Plotting Histogram.

L1: Histogram plotted for the Intra Hamming distance calculated between selected device at Fabrication house and same selected device at Field.

L2: Histogram plotted for the Inter Hamming distance calculated between selected device at Fabrication house and other device at Field.

Algorithm-2 with 8-bits response

Algorithm-2 is evaluated by taking 8-bits response and histogram is plotted as shown in figure 4.6. This figure shows that intra hamming distance and inter hamming distance bars overlap each other in 8 bits response.

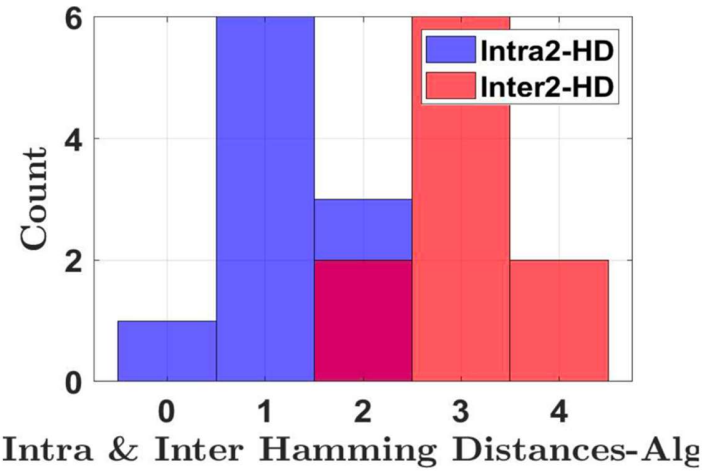


Figure 4.6 | Algorithm 2- Inter & Intra Hamming Distances for *B* = 8-bits.

Algorithm-2 with 16-bits response

Algorithm-2 is evaluated by taking 16-bits response and histogram is plotted as shown in figure 4.7. This figure shows approximately 5 bits difference randomly between intra hamming distance and inter hamming in 16 bits response.

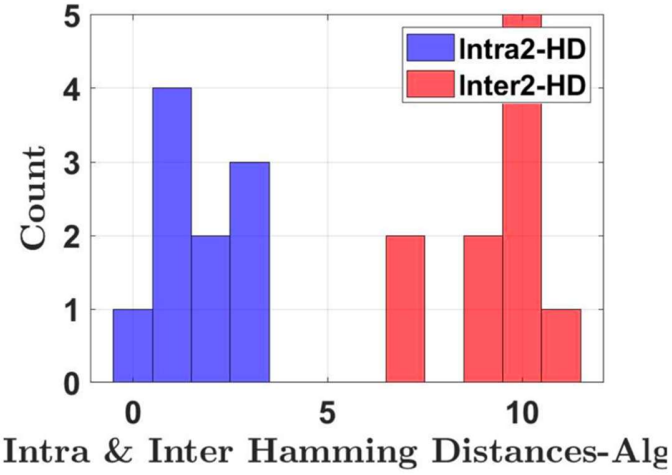


Figure 4.7 | Algorithm 2-Inter & Intra Hamming Distances for *B* = 16-bits.

Algorithm-2 with 32-bits response

Algorithm-2 is evaluated by taking 32-bits response and histogram is plotted as shown in figure 4.8. This figure shows that intra hamming distance and inter hamming distance bars overlap each other in 32 bits response.

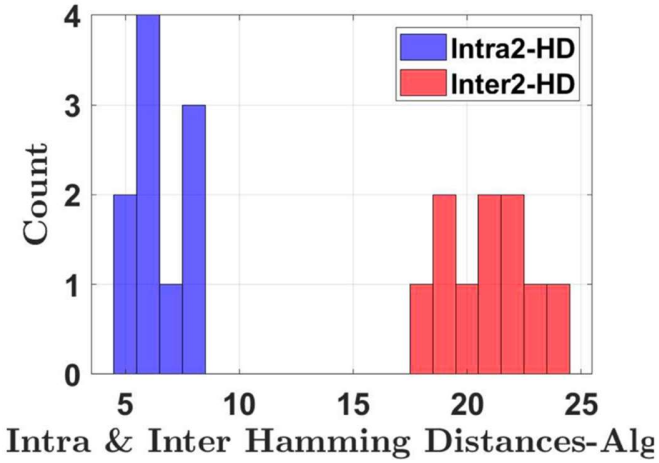


Figure 4.8 | Algorithm 2-Inter & Intra Hamming Distances for *B* = 32-bits.

Algorithm-2 with 64-bits response

Algorithm-2 is evaluated by taking 64-bits response and histogram is plotted as shown in figure 4.9. This figure shows approximately 2 bits difference between intra hamming distance and inter hamming in 64 bits response. Only 2 bits difference makes it difficult to identify the valid and fake device.

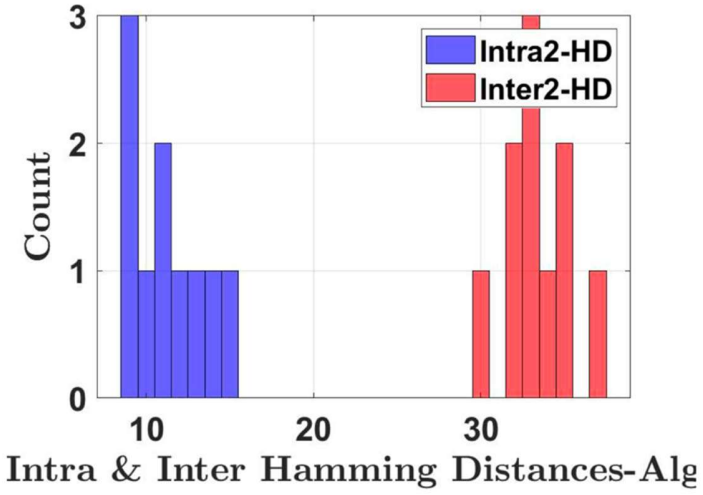


Figure 4.9 | Algorithm 2-Inter & Intra Hamming Distances for *B* = 64-bits.

Algorithm-2 with 128-bits response

Algorithm-2 is evaluated by taking 128-bits resonse and histogram is plotted as shown in figure 4.10. This figure shows approximately 20 bits difference between intra hamming distance and inter hamming in 128 bits response. Valid and fake device can be clearly identified with this figure of 128 bits response.

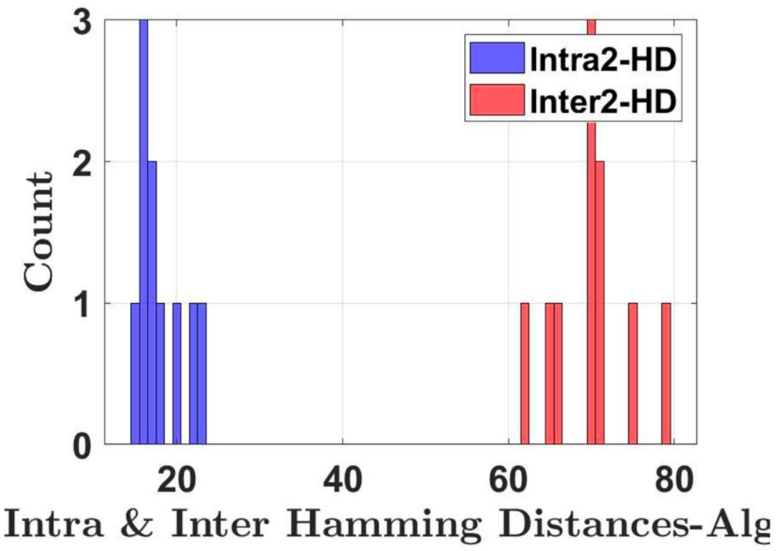


Figure 4.10 | Algorithm 2-Inter & Intra Hamming Distances for *B* = 128-bits.

4.3.3 Algorithm-3

We have observed in algorithm 1 and 2 that to identify the valid and fake device larger challenge input bits (128 bits) are required. Algorithms 1 and 2 cannot identify the valid and fake device clearly up to 64 bits challenge input.

The purpose of Algorithm-3 is to eliminate thermal noise of devices at field by repeating the steps to obtain the golden responses. One device is selected for simulation at fabrication house called a Fabrication device and the same device at field is called a valid device. One other device is called a fake device. Algorithm-3 generates Golden response of Fabrication device, Valid response and Fake response.

Table 4.6 illustrates the Algorithm-3, Response generation will be same as algorithm-1&2 and Golden Response is generated at using golden response function and Hamming distance is calculated using Hamming distance function.

Require: Valid_response, Fake_response, Fab_Golden_response
1: Valid_Golden_response= **Golden_response** (Valid_response) *% Golden response generation of selected device at field.*
2: Fake_Golden_response= **Golden_response** (Fake_response) *% Golden response generation of other device at field.*
3: Intra3_HD = **Hamming_distance** (Fab_Golden_response, Valid_Golden_response) *%Hamming distance between Fabrication golden response and valid golden response.*
6: Inter3_HD = **Hamming_distance** (Fab_Golden_response, Fake_Golden_response) *% Hamming distance between Fabrication golden response and fake golden response.*
7: **Return** diff= Inter3_HD – Intra3_HD

Table 4.9 | Pseudocode for Algorithm-3.

L1: Golden Response generated for selected device at Field.

L2: Golden response generation of other device at field.

L3: Intra Hamming distance calculated between Fabrication golden response and valid golden response

L4: Inter Hamming distance calculated between Fabrication golden response and fake golden response.

L5: Difference of Inter and Intra Hamming distance calculated.

Table 4.10| Algorithm-3 Inter, Intra Hamming Distances & Differences for *B* = 8, 16, 32, 64, 128-bits.

BITS	8	16	32	64	128
INTRA3	0.1250	0.1250	0.1875	0.1719	0.1328
INTER3	0.2500	0.8750	0.5626	0.5469	0.4062
DIFF3	1	12	12	24	35

Table 4.7 illustratess that there is a 1 bit distance between Intra HD and Inter HD for response bits size 8. This distance between Intra HD and Inter HD increases as the bit size increases and this distance reaches up to 35 bits for 128 bits. This Algorithm 3 indicates that it can identify the difference between valid and fake device very easily even for the smaller bit size.

CHAPTER 5

PERFORMANCE EVALUATION

This chapter illustrates the performance of Arbiter PUF when the three authentication algorithms are applied. The simulations were conducted using MATLAB version R2021b running on Processor 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz, 1690 Mhz, 2 Core(s), 4 Logical Processor(s).

Random number generation used the built-in function randn and rand. Simulations of Arbiter were done for a given value of the response size B. For each value of B a new simulation was performed and the simulations were run 1,000 times to obtain the mean values of the performance graphs.

The performance results for the arbiter PUF are summarized in below Table 4.1 for the three authentication algorithms. We have considered the following parameter values Number of switching stages S as 4, Signal to Noise Ratio (SNR)in DB as 30 dB, Standard deviation due to random process variation (RPV) as 0.4, and Setup & Hold time of the flipflop is considered as 0.3 to 0.7.

The performance of Arbiter PUF in the table 5.1 indicates that in Algorithm 1, Intra HD and Inter HD overlap each other for 8 bits to 32 bits which makes it difficult to identify the valid and fake device. To identify the Valid and fake device in algorithm-1, response size to be increase to 128bits.

For Algorithm 2, no gap between Intra HD and Inter HD for 8 bits and 16 bits that means performance of valid and fake device are very similar to each other which makes it difficult to clearly identify the difference between valid and fake device. Distance of 1 bit started from 32 bits and this distance increased to 13 bits for 128 bits. Same as algorithm-1, to identify the Valid and fake device response size to be increase to 128bits.

Both the algorithms 1 and 2 are not suitable for lower size bits. These algorithms 1 and 2 can only be used for higher size bits.

For proposed Algorithm 3, there is a 1 bit distance between Intra HD and Inter HD for response bits size 8. This distance between Intra HD and Inter HD increases as the bit size increases and this distance reaches up to 35 bits for 128 bits. This Algorithm 3 indicates that it can identify the difference between valid and fake device very easily even for the smaller bit size.

Table 5.1 | Algorithm-1, 2, 3 Inter and Intra Hamming Distances for B = 8, 16, 32, 64, 128-bits.

	B (bits)	8	16	32	64	128
ALGORITHM #1	INTRA HD	0.2629	0.0710	0.1385	0.1728	0.2084
	INTER HD	0.6314	0.4854	0.4811	0.5715	0.4876
	DIFFERENCE	-3	0	-2	10	5
ALGORITHM #2	INTRA HD	0.2567	0.0788	0.1253	0.1610	0.1965
	INTER HD	0.8090	0.4464	0.4795	0.5758	0.4854
	DIFFERENCE	0	0	1	12	13
ALGORITHM #3	INTRA HD	0.1250	0.0625	0.0625	0.1562	0.2266
	INTER HD	0.7500	0.5625	0.4375	0.6094	0.4609
	DIFFERENCE	5	8	12	29	30

CHAPTER 6

CONCLUSION

Arbiter PUFs were reviewed with its structure and operation. Statistical models for this PUF and the random variable was identified. The main system parameters were also discussed in the developed models. Techniques used by the manufacturer to obtain the device CRP data were discussed. This work then reviewed three authentication algorithms to obtain the CRP. The performance of Arbiter PUF is presented and it is concluded that algorithm 3 gives the best performance as it clearly authenticates the valid device and identifies the valid and fake device. The rate of error in the identification problem significantly reduced.

REFERENCES

[1] *"Chapter 3 - An introduction to cryptoGraphy," in Next Generation SSH2 Implementation, D. Liu, M. Caceres, T. Robichaux, D. V. Forte, E. S. Seagren, D. L. Ganger, B. Smith, W. ayawickrama, C. Stokes, and J. Kanclirz, Eds. Syngress, pp. 41–64. ISBN 978-1-59749-283-6.*

[2] *T. Unterluggauer and S. Mangard, "Exploiting the physical disparity: Side-channel attacks on memory encryption," in Constructive SideChannel Analysis and Secure Design, ser. Lecture notes in Computer Science, F.-X. Standaert and E. Oswald, Eds. Springer International Publishing. ISBN 978-3-319-43283-0 pp.*

[3] *D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 10, pp. 1200– 1205, Oct 2005. doi: 10.1109/TVLSI.2005.859470*

[4] *B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in Proceedings of the 9th ACM conference on Computer and communications security. ACM, 2002*

[5] *J. W. Lee, Daihyun Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525), June 2004. doi: 10.1109/VLSIC.2004*

[6] *M. Barbareschi, "Notions on silicon physically unclonable functions," in Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment, N. Sklavos, R. Chaves, G. Di Natale, and F. Regazzoni, Eds. Springer International Publishing*

[7] Gebali, Fayez; Mamun, Mohammad, *Review of physically unclonable functions (PUFs): structures, models, and algorithms*
NRC Publications Archive Archives des publications du CNRC https://doi.org/10.3389/fsens.2021.751748 Frontiers in Sensors, 2, pp. 1-13, 2022-01-11

[8] Fayez Gebali, *Performance Analysis of Arbiter Physically Unclonable Functions* received as private communications on March 2022.

[9] *B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in Proceedings of the 9th ACM conference on Computer and communications security - CCS '02, 2002*

[10] *Making Use of Manufacturing Process Variations: A Dopingless Transistor Based-PUF for Hardware-Assisted Security Venkata P. Yanambaka, Saraju P. Mohanty, Elias Kougianos, IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING, VOL. XX, NO. Y, MONTH 2018*

[11] *S. Hemavathy and V. S. K. Bhaaskaran, "Arbiter PUF—A Review of Design, Composition, and Security Aspects," in IEEE Access, vol. 11, pp. 33979-34004, 2023, doi: 10.1109/ACCESS.2023.3264016.*

[12] *Kulkarni, S., Vani, R.M., Hunagund, P.V. (2021). Designing of Arbiter PUF for Securing IP and IoT Devices. In: Jeena Jacob, I., Kolandapalayam Shanmugam, S., Piramuthu, S., Falkowski-Gilski, P. (eds) Data Intelligence and Cognitive Informatics. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-8530-2_10*

[13] *J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in CryptoGraphic Hardware and Embedded Systems- CHES 2007, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007*

[14] *A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions," in Embedded systems design with FPGAs. Springer, 2013*

[15] *A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). IEEE, 2010*

[16] *NIST Special Publication 800-22. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Information Technology Laboratory of the National Institute of Standards and Technology, May 2000.*

[17] *J. Lin, "Divergence measures based on the Shannon entropy," in IEEE Transactions on Information Theory, vol. 37, no. 1, pp. 145-151, Jan. 1991, doi: 10.1109/18.61115.*

[18] *A. Tsuneda and K. Morikawa, "A study on random bit sequences with prescribed auto-correlations by post-processing using linear feedback shift registers," 2013 European Conference on Circuit Theory and Design (ECCTD), 2013, pp. 1-4, doi: 10.1109/ECCTD.2013.6662328.*

[19] *V. B. Suresh and W. P. Burleson, "Entropy extraction in metastability-based TRNG," 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2010, pp. 135-140, doi: 10.1109/HST.2010.5513099.*

[20] *O'Donnell, Charles W., G. Edward Suh and Srinivas Devadas. "PUF-Based Random Number Generation." (2004). MIT CSAIL CSG Technical Memo 481.*

APPENDIX A

Matlab code for simulating an Arbiter PUF for Algorithm-1, Algorithm-2 and Algorithm-3.

```
clear;
close all;
clc;
%% Arbiter system parameters
S = 4; % Number of Stages
t_setup = 0.5; % Setup & Hold time of the flipflop
mu_p = 1.2; % Mean of random process variation (RPV)
sigma_p = 0.4; % Standard deviation due to random process variation (RPV)
mu_n = 0; % Mean of CMOS noise
SNR=30; % Signal to Noise Ratio (SNR)in DB
N=1000; % Number of Iteration
B= [8,16,32,64,128];% Number of Response bits
Fab_response=zeros(B,N);
Valid_response=zeros(B,N);
Fake_response=zeros(B,N);
Fab_flag=zeros(B,N);
Valid_flag=zeros(B,N);
Fake_flag=zeros(B,N);
%% Define sigma_n Standard deviation due to CMOS noise
sigma_n=f_sig(sigma_p,mu_p,SNR);
%% Define RPV Delay Matrices for four paths all B rows and S stages
T_Fab_pUU = f_T(sigma_p,mu_p,B,S); % RPV Delay from Upper Input to Upper Output
T_Fab_pLL = f_T(sigma_p,mu_p,B,S); % RPV Delay from Lower Input to Lower Output
T_Fab_pLU = f_T(sigma_p,mu_p,B,S); % RPV Delay from Lower Input to Upper Output
T_Fab_pUL = f_T(sigma_p,mu_p,B,S); % RPV Delay from Upper Input to Lower Output
%% Define RPV Delay Matrices for four paths all B rows and S stages
T_Fake_pUU = f_T(sigma_p,mu_p,B,S); % PV Delay from Upper Input to Upper Output
T_Fake_pLL = f_T(sigma_p,mu_p,B,S); % PV Delay from Lower Input to Lower Output
T_Fake_pLU = f_T(sigma_p,mu_p,B,S); % PV Delay from Lower Input to Upper Output
T_Fake_pUL = f_T(sigma_p,mu_p,B,S); % PV Delay from Upper Input to Lower Output
%% Challege
c=randi(2,1,S)-1;
%% For Iterations
for n=1:N
%% Define Noise Delay values for four paths all B rows and S stages
T_Fab_nUU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Upper Output
T_Fab_nLL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Lower Output
T_Fab_nLU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Upper Output
T_Fab_nUL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Lower Output
%% Define Noise Delay values for four paths all B rows and S stages

T_Valid_nUU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Upper Output
T_Valid_nLL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Lower Output
T_Valid_nLU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Upper Output
T_Valid_nUL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Lower Output
%% Define Noise Delay values for four paths all B rows and S stages
T_Fake_nUU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Upper Output
```

```
T_Fake_nLL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Lower Output
T_Fake_nLU = f_T(sigma_n,mu_n,B,S); % Noise Delay from Lower Input to Upper Output
T_Fake_nUL = f_T(sigma_n,mu_n,B,S); % Noise Delay from Upper Input to Lower Output
%% Fabrication Device Total Delay of upper and lower output at final stage
[Fab_delay_upper, Fab_delay_lower] = ...
f_Cumulative_delay(c,T_Fab_pUU,T_Fab_pUL,T_Fab_pLL,T_Fab_pLU,T_Fab_nUU,
T_Fab_nUL,T_Fab_nLL,T_Fab_nLU);
%% Valid Device Total Delay of upper and lower output at final stage
[Valid_delay_upper, Valid_delay_lower] = ...
f_Cumulative_delay(c,T_Fab_pUU,T_Fab_pUL,T_Fab_pLL,T_Fab_pLU,T_Valid_nUU,
T_Valid_nUL,T_Valid_nLL,T_Valid_nLU);
%% Fake Device Total Delay of upper and lower output at final stage
[Fake_delay_upper, Fake_delay_lower] = ...
f_Cumulative_delay(c,T_Fake_pUU,T_Fake_pUL,T_Fake_pLL,T_Fake_pLU,T_Fake_nUU,
T_Fake_nUL,T_Fake_nLL,T_Fake_nLU);
%% Function for Generate Responses of Fabrication Device
[Fab_response(:,n),Fab_flag(:,n)] = f_response (t_setup,Fab_delay_upper,
Fab_delay_lower);
%% Function for Generate Responses for Valid Device
[Valid_response(:,n),Valid_flag(:,n)] = f_response (t_setup,Valid_delay_upper,
Valid_delay_lower)
%% Function for Generate Responses for Fake Device
[Fake_response(:,n),Fake_flag(:,n)] = f_response (t_setup,Fake_delay_upper,
Fake_delay_lower);
end
%% Alogrithm #1
% Function for Hamming Distance Calculations
[Intra1_HD] = f_HD (Fab_response,Valid_response);
[Inter1_HD] = f_HD (Fab_response,Fake_response);
% Normalization of Hamming Distance
Intra1 = mean(Intra1_HD)/B
Inter1 = mean(Inter1_HD)/B
max1=max(Intra1_HD);
min1=min(Inter1_HD);
diff1=(min1-max1)
% Plot Alogrithm #1

hold on
histogram(Intra1_HD,'FaceColor','b')
xlabel('\textbf{Intra \& Inter Hamming Distances-Alg1}','interpreter',
'latex','FontSize',20,'FontWeight','bold')
ylabel('\textbf{Count}','interpreter', 'latex','FontSize',20,'FontWeight','bold')
set(gca,'FontSize',10,'FontWeight','bold' )
histogram(Inter1_HD, 'FaceColor','r')
box on
grid on
legend('Intra1-HD', 'Inter1-HD')
%% Alogrithm #2
% Function to create golden response
[golden_response,golden_flag]= f_golden_response(Fab_response);
% Function for Hamming Distance Calculations
[Intra2_HD] = f_HD (golden_response,Valid_response);
[Inter2_HD] = f_HD (golden_response,Fake_response);
```

```
% Normalization of Hamming Distance
Intra2 = mean(Intra2_HD)/B;
Inter2 = mean(Inter2_HD)/B;
max2=max(Intra2_HD);
min2=min(Inter2_HD);
diff2=(min2-max2);
% Plot Alogrithm #2
histogram(Intra2_HD, 'FaceColor','b')
xlabel('\textbf{Intra \& Inter Hamming Distances-Alg2}','interpreter',
'latex','FontSize',20,'FontWeight','bold')
ylabel('\textbf{Count}','interpreter', 'latex','FontSize',20,'FontWeight','bold')
set(gca,'FontSize',20,'FontWeight','bold' )
hold on
histogram(Inter2_HD, 'FaceColor','r')
box on
grid on
legend('Intra2-HD', 'Inter2-HD')
%% Alogrithm #3
% Function to create golden response
[golden_Valid_response,golden_Valid_flag]= f_golden_response(Valid_response)
[golden_Fake_response,golden_Fake_flag]= f_golden_response(Fake_response)
% Function for Hamming Distance Calculations

[Intra3_HD] = f_HD (golden_response,golden_Valid_response);
[Inter3_HD] = f_HD (golden_response,golden_Fake_response);
Intra3=Intra3_HD/B;
Inter3=Inter3_HD/B;
diff3= Inter3_HD-Intra3_HD
```

APPENDIX B

Matlab function for calculating the standard deviation (sigma).

```
function [sigma_n] = f_sig(sigma_p,mu_p,SNR)

   % mu_p Mean of random process variation (RPV)
   % sigma_p Standard deviation due to random process variation (RPV)
   % Singal to Noise Ratio (SNR)in DB
   % sigma_n Standard deviation due to CMOS noise
   sigma_n =sqrt((sigma_p^2+mu_p^2)/10^(SNR/10));
end
```

APPENDIX C

Matlab function for generating the PUF response from the time delays of lower and upper arm of an arbiter PUF.

```
function [r,flag] = f_response (t_setup,Total_delay_upper,Total_delay_lower)

  %% Function for Response Generation
  %t_setup is setup and hold time
  B=length(Total_delay_upper);
  diff=zeros(1,B); %Initialization of Delay Difference array.
  r=zeros(B,1); %Initialization of response array.
  flag=zeros(B,1); % Initialization of flag array.
  for b = 1:B
  diff(b)=Total_delay_upper(b)-Total_delay_lower(b);
  abs_diff=abs(diff(b));
  if abs_diff> t_setup % delay difference more thean setup/hold time
  if diff(b)>0
  r(b) = 0;
  else
  r(b) = 1;
  end
  else
  flag(b)=1; % delay difference less than setup/hold time
  probability=rand;
  if probability<0.5
  r(b)=1;
  else
  r(b)=0;
  end
  end
end
```

APPENDIX D

Matlab function for calculating the cumulative delay from the lower and upper arm delay of an arbiter PUF.

```
function [Total_delay_upper, Total_delay_lower] = ...

  f_Cumulative_delay(c,T_pUU,T_pUL,T_pLL,T_pLU,T_nUU,T_nUL,T_nLL,T_nLU)
  Total_delay_upper= 0;
  Total_delay_lower= 0;
  for i=1:length(c)
  if c(i)==0
  Total_delay_upper = Total_delay_upper+T_pUU(:,i)+T_nUU(:,i); %Column Vector
  of B components
  Total_delay_lower = Total_delay_lower+T_pLL(:,i)+T_nLL(:,i); %Column Vector
  of B components
  else
  Total_delay_upper = Total_delay_upper+T_pLU(:,i)+T_nLU(:,i); %Column Vector
  of B components
  Total_delay_lower = Total_delay_lower+T_pUL(:,i)+T_nUL(:,i); %Column Vector
  of B components
  end
end
```

APPENDIX E

Matlab function for calculating the hamming distance from the responses of an arbiter PUF.

```
function [HD] = f_HD (Fab_response,Device_response)

  HD = sum(abs(Fab_response-Device_response));
end
```