Fall 12-1-2016

Model Building and Security Analysis of PUF-Based Authentication

Wenjie Che
University of New Mexico

Wenjie Che

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Professor Jim Plusquellic , Chairperson

Professor Manel Martinez-Ramo

Dr. Fareena Saqib

Dr. Jim Aarestad

Model Building and Security Analysis of PUF-Based Authentication

by

Wenjie Che

B.E., Telecommunications Engineering, Hunan Normal University, 2010

M.E. Computer Technology, Hunan University, 2013

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2016

Dedication

This dissertation is dedicated to my wife Junlan Shi and my family, for all the love, support, and the many sacrifices they made.

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Jim Plusquellic for his untiring assistance, guidance, encouragement at every stage of this study. He is the main source of inspiration and encouragement throughout my whole graduate study. I have benefited and gained a lot working with him in both science and engineering fields. Without his great support and guidance I wouldn't have been able to come this far.

My deepest appreciation and thanks go to Porf. Manel Martinez-Ramon for his guidance and valuable discussions in the machine-learning area. I would like to thank Prof. Payman Zarkesh-Ha for his support and guidance. My special thanks go to my colleague Dr. Dylan Ismari, from whom I learned a lot during my Ph.D study. I am grateful to all my group members for all their support and encouragement, they are Dr. Fareena Saqib, Goutham Pocklassery, Venkata Kishore Kajulur, Akshay Sudhir Vaidya, Ian Wilcox, Mitchell Martin. My sincere thanks also go to all the fellows in the ECE department of UNM, for their assistance, motivation and valuable comments to improve the dissertation.

Model Building and Security Analysis of PUF-Based Authentication

by

Wenjie Che

B.E., Telecommunications Engineering, Hunan Normal University, 2010

M.E. Computer Technology, Hunan University, 2013

Ph.D, Engineering, University of New Mexico, 2016

ABSTRACT

In the context of hardware systems, authentication refers to the process of confirming the identity and authenticity of chip, board and system components such as RFID tags, smart cards and remote sensors. The ability of physical unclonable functions (PUF) to provide bitstrings unique to each component can be leveraged as an authentication mechanism to detect tamper, impersonation and substitution of such components. However, authentication requires a strong PUF, i.e., one capable of producing a large, unique set of bits per device, and, unlike secret key generation for encryption, has additional challenges that relate to machine learning attacks, protocol attacks and constraints on device resources. We describe the requirements for PUF-based authentication, and present a PUF primitive and protocol designed for authentication in resource constrained devices. Our experimental results are derived from a 28 nm Xilinx FPGA.

A special class of Physical Unclonable Functions (PUFs) referred to as strong PUFs can be used in novel hardware-based authentication protocols. Strong PUFs are required for authentication because the bitstrings and helper data are transmitted openly by the token to the verifier and therefore, are revealed to the adversary. This enables the adversary to carry out attacks against the token by systematically applying challenges and obtaining responses in an attempt to machine-learn and later predict the token's response to an arbitrary challenge. Therefore, strong PUFs must both provide an exponentially large challenge space and be resistant to machine-learning attacks in order to considered secure. We investigate the security properties of a Hardware-embedded Delay PUF called HELP which leverages within-die variations in path delays within a hardware-implemented macro (functional unit) as a random source of information for bitstring generation. Several features of the HELP processing engine significantly improve its resistance to model-building attacks. Most important is a novel linear transformation proposed within the HELP processing engine for dealing with changes in delay introduced by adverse temperature-voltage (environmental) variations. The technique also increases entropy by making the measured path delay values dependent on the other values included in the distribution used to generate the entire bitstring.

Statistical properties including uniqueness, randomness and reproducibility are commonly used as metrics for Physical Unclonable Functions (PUFs). When PUFs are used in authentication protocols, the first two metrics are critically important to the overall security of the system. Authentication reveals the bitstrings (and helper data if used) to the an adversary, and makes the PUF vulnerable to tactics that can lead to successful cloning and impersonation. We investigate security metrics including Entropy, uniqueness and randomness using hardware data collected from a set of 45 Xilinx Zynq FPGAs which implements HELP. A novel technique is proposed that allows the verifier to randomly or purposefully offset path delays to obfuscate (in the former case) and/or tune (in the latter case) the bitstring generation process. We show that tuning additionally has a significant impact on the statistical quality of the bitstrings.

Stability across environmental variations such as temperature and voltage, is critically important for Physically Unclonable Functions (PUFs). Nearly all existing PUF systems to date need a mechanism to deal with “bit flips” when exact regeneration of the bitstring is required, e.g., for cryptographic applications. Error correction (ECC) and error avoidance schemes have been proposed but both of these require helper data to be stored for the regeneration process. Unfortunately, helper data adds time and area overhead to the PUF system and provides opportunities for adversaries to reverse engineer the secret bitstring. We propose a non-volatile memory-based (NVM) PUF that is able to avoid bit flips without requiring any type of helper data. We describe the technique in the context of emerging nano-devices, in particular, resistive random access memory (Memristor) cells, but the methodology is applicable to any type of NVM including Flash.

Table of Contents

List of Figures

CHAPTER 1
Introduction

Physically Unclonable Functions (PUFs) are emerging as an alternative to conventional approaches to storing secret keys in ICs. PUFs extract entropy from variations in the physical and electrical properties of ICs, that are unique to each IC, as a means of generating secrets. Secret keys derived from PUFs mitigate the vulnerabilities of embedded digital keys mainly in two ways: (1) the generated secret keys are volatile and only present in a digital form when the chip is powered on and running. thus making it difficult for adversaries to steal by invasive attacks; (2) even if the secret key is known, it is intractable for the manufacturer to duplicate a second chip with an identical key. These secrets can be used in various security applications including device identification, authentication, metering, remote activation and encryption [1]. This dissertation is mainly focused on analyzing the entropy source and the security strength of the hardware-embedded delay PUF (HELP) in the context of authentication.

1.1 Metrics for PUFs

There are typically three major metrics for measuring the quality of a PUF: Uniqueness, reliability and randomness.

1.1.1 Uniqueness

Uniqueness refers to how unique are the responses generated by the PUF from different chips when they are applied with the same challenge. The better uniqueness of a

PUF, the more difficult for an adversary to copy an existing PUF instance that possesses similar Challenge-Response Pair (CRP) behaviors, the more desirable for the responses to be used as chip identifiers. A common method to quantify the uniqueness metric of a PUF is to compute the average inter-chip Hamming Distance (inter-HD) for a group of chips on which a particular PUF instance is embedded. For a given challenge, the inter-HD between two PUF instances is defined as the number of different bits that exist between the two PUF responses. Inter-HD can also be represented as a fraction which is calculated by dividing the number of different bits by the total number of compared bits. Assume the PUF response is n bits, the average inter-HD for a groups of m chips (PUF instances) can be computed using the following formula [7]:

$$Inter\_HD_{ave} = \frac{\sum_{u=1}^{m-1} \sum_{v=u+1}^{m} HD(R_u, R_v)}{n} \times 100\% \tag{1.1}$$

where $HD(R_u, R_v)$ represents the hamming distance between the responses generated by the u-th and v-th chips for a given challenge.

An ideal value for the average inter-HD for a group of chips is 50%, which indicates the best uniqueness is achieved when half of the bits are different between any two compared responses on average.

1.1.2 Reliability (Reproducibility)

The uniqueness metric is used to measure the quality of a PUF whether the responses generated by different PUF instances are unique enough to distinguish between chips. For the same PUF instance, however, it is desirable that the responses should always keep constant when the same challenge are applied multiple times under different environmental conditions. This is true because in either authentication or secret key generation, we expect that the original secrets can be reliably reproduced by the PUF whenever proof-of-evidence needs to be provided later. We use the reliability (or reproducibility) metric to measure how reliable it is to regenerate the original responses for a PUF. Similar to uniqueness, reliability can also be measured by calculating the average Intra-chip Hamming Distance (Intra-HD). For a particular PUF instance, intra-HD refers to the Hamming Distance between the two measured responses when the same challenge is applied twice. Assume x response samples are measured under different conditions (different temperatures or supply voltages) for a chip i, the average intra-HD can be calculated as [7]:

$$Intra\_HD_{ave} = \frac{1}{x}\sum_{y=1}^{x} HD\frac{\left(R_i, R'_{i,y}\right)}{n} \times 100\% \qquad (1.2)$$

where R_i represents the response measured under normal temperature and voltage conditions (nominal condition), $R'_{i,y}$ stands for the y-th measured response sample of R_i. Note intra-HD is calculated from the responses generated from the same chip for the same challenge, therefore, the smaller value it is, the better reliability is achieved.

The ideal value of the intra-HD is 0%, which implies that the original responses can be exactly reproduced without any error bits across all the sampled conditions. However, the ideal reliability can hardly be achieved due to the sensitivity of PUF responses to varying environmental conditions. Typically, a PUF response bit is generated depending on the sign of the difference value for two compared entropy sources, e.g., a “1” is generated for a positive sign and a “0” otherwise. Unfortunately, the entropy source leveraged in PUFs are analog in nature and hence can easily be impacted by varying environmental conditions, e.g., temperatures or supply voltage noise (or TV noise). For those response bits that were generated from two close entropy values, it is very likely the two original entropy values will vary under environmental noise and jump across each other, resulting in error response bits or “bit flips”. Such bit flips directly cause degradation to the reliability performance of a PUF. In applications like authentication, a small portion of bit flips within the regenerated bitstring (secrets) can be tolerated. This is because as long as the majority of the reproduced bits are consistent with the original one, the chip can still be authenticated as genuine or fake. In cryptographic scenarios such as key generation, however, every single bit of the secret bitstring are required to be accurately regenerated, since even one single bit flip in the secret key will generate a totally different result after the key is applied to the cryptographic engine. In order to improve reliability of a PUF, different methods have been proposed to either correct or avoid bit flips, which will be discussed in the subsection later.

1.1.3 Randomness

Randomness is another major metric that measures how random the the generated responses are by a PUF. A random bit sequence can be interpreted as the result of a sequence of “flips” of an unbiased coin [8]. The randomness of the generated responses can be evaluated using statistical test suite like NIST test [8]. The NIST test suite is regarded as the industrial standard to test random bit sequence for cryptographic applications. NIST consists of 15 subtests which covers the uniformity, correlation and approximate entropy tests for the bit sequence.

1.2 Helper data for PUFs

As mentioned in section 2.1.2, entropy sources extracted from PUFs are subject to ambient environmental variations, which are likely to cause inevitable bit flips within the regenerated responses. To deal with this reliability issue, two classes of methods have proposed to either correct the error bits or to avoid bit flips: Error correction code (ECC) and thresholding technique.

1.2.1 Error Correction Schemes

The most popular method is to use error correction information (ECC) for bit flips correction. Two phases are involved in the ECC scheme: the first time of generating the secret bitstring is called the enrollment process, and regenerating the same secret bitstring later is the regeneration process. ECC works in a fashion that the error correcting information, commonly a syndrome, is extracted from the secret bitstring during the enrollment process, which is later used to correct error bits during regeneration. The error correction information or syndrome is public information and also referred to as helper data and is stored in a reliable storage device, e.g., an on-chip non-volatile memory or off-chip storage. However, there are several drawbacks that come along with the helper data. A main security concern is that such helper data reveals information of the secret bitstrings and thus makes the secrets vulnerable to be stolen by attackers [5]. Besides, extra overheads are incurred to the design due to implementing the syndrome and storing the helper data. Many error correction schemes have been summarized in [5], but most of them require fairly heavyweight error correction logic, and they inevitably leak information of the secret bits.

Error correction techniques can generally be categorized into three classes [9]: Code-Offset [10][11][3], Index-Based Syndrome [12][13][14] and Pattern Vector [17]. In Code-Offset schemes, the entropy loss of the Syndrome bits is attributed to the maximum number of exposed Syndrome bits. In this sense, the syndrome bits should be as small as possible so that the leaked information by the exposed bits does not exceed the entropy contained within the secret bitstring. The entropy within the PUF response can be measured by the min-entropy of PUF outputs used, hence the remaining entropy after exploiting a Code-Offset scheme depends on the min-entropy of the PUF outputs and the number of Syndrome bits. The basic idea of the Index-Based Syndrome scheme is to use the divided syndrome outputs (in unit of words) as the index of a sequence of PUF outputs. The scheme is proven to be theoretically secure since bias within the PUF outputs is not expected to be amplified, hence leaking no information of the min-entropy of the secret bits. However, it is only true under the assumption that the PUF outputs have to be independent and identically distributed (i.i.d), which is difficult to measure in practical scenarios. It is further analyzed in [14] that there are two-out-of-three bits information leakage without this assumption.

The third class is referred to as the Pattern Vector scheme which reverses the paradigm routine of conventional ways of generating secrets using a PUF. Instead of fixing a set of challenges and using their corresponding responses as secrets, Pattern Vector records a set of response sequences as public helper information during the provisioning process, which is used later as a set of patterns to be compared with the regenerated response bits. If an approximate match happens, then the index value of the challenge that generates this matched response will be used as a sub-key. Multiple rounds of such matching process during regeneration comprises the final key. The security of the scheme relies on the fact that the challenges of the exposed response pattern set (helper data) are not knowledgeable to adversaries. Since only a small set of responses are exposed, the authors claimed that it is infeasible for attackers to successfully build a PUF model with a limited number of CRPs. Moreover, the scheme is lightweight for implementation since it eliminates the need for complex error correction logic like BCH decoders.

However, authors in [6] proposed an attacking frame that is able to recover full keys from the Pattern Vector scheme by manipulating the helper data. An attacker is able to gradually retrieve the full bitstreams by statistically observing the failure rates of Pattern Vector generators, thus eventually recover the secret indexes.

1.2.2 Thresholding Technique

Error correction schemes (ECC) are designed to correct error bits within the regenerated secret bitstrings due to ambient environmental variations. An alternative scheme to ECC can be referred to as the thresholding technique, which have been proposed to avoid bit flips during regeneration. As introduced in previous sections, a response bit is typically generated depending on the polarity of the difference between two compared entropy values. It is easy to speculate that a polarity is more likely to flip in cases where the difference of the two compared values is too tiny to resist to fluctuations caused by varying environmental conditions. In this regard, response bits derived from such cases are called “weak” bits, while those bits whose entropy pair possesses large difference gaps are “strong” bits. Thresholding techniques are proposed to discard those “weak” bits during enrollment so that only strong response bits are selected for regeneration. This bit flips avoiding process is achieved by setting a “threshold” which partitions all the possible entropy pairings into two classes: “weak” pairings whose differences are smaller than the “threshold” and “strong” pairings whose difference exceeds the “threshold”. Then the positioning information for these “weak” pairings are recorded in a public storage as “helper data”, which will be referenced later during regeneration to indicate which pairings should be discarded.

Although thresholding techniques have been proposed in various types of PUF designs, they share the same idea as is illustrated above. An early version of applying this idea is named as the “masking scheme”, which is proposed in [15] for Ring Oscillator (RO) PUFs. The scheme uses a masking vector that selects which RO pairs should be picked up for bitstring regeneration. A typical example is the “1-out-of-k” scheme where the pair that has the maximum frequency distance among all the k pairing is selected. Another version of thresholding technique is proposed in [16] for the Power-grid (PG) PUF. The entropy source of the PG-PUF is the resistance variations that exist in the power grid metal wires and transistors. Resistance can be measured by equivalent voltage drops across entropy cells by creating a voltage divider. After measuring the noise level caused by varying measurement and environmental conditions, a threshold voltage value is selected which is defined as the upper bound on the voltage differences of two compared entropy cells. A similar voltage thresholding scheme is proposed in [18] for a transmission gate PUF (or TG PUF) to ensure reliably regenerating PUF responses. Recently, the thresholding idea is used for a Sense Amplifier based PUF (SA-PUF) to select SA pairs that are able to generate reliable bits.

Compared to error correction schemes, the thresholding technique does not require complex implementation logic for error correction codes and are generally accepted that they are not vulnerable to information leakage. However, “helper data” information that records the positioning information of reliable bits need to be stored, which also incurs inevitable overheads. Moreover, a recent survey [19] on helper data algorithms have disproved the assumption that bit selection schemes (thresholding) have no leakage.

1.3 Organization

The dissertation is organized as follows: chapter 2 is based on the paper “A Non-Volatile Memory Based Physically Unclonable Function without Helper Data”, which is focused on addressing the reliability issue of PUFs. Chapter 3 is based on the paper “PUF-Based Authentication”, which analyzes the entropy introduced from multiple sources and proposed an authentication protocol that is based on the HELP PUF. Chapter 4 is based on a paper that is titled “Leveraging Distributions in Physical Unclonable Functions”, which investigates the enhanced robustness introduced by the linear transformation within the HELP processing engine against model-building. Chapter 5 is based on a paper under review “A Novel Offset Method for Improving Bitstring Quality of a Hardware-Embedded Delay PUF”, which proposes an offset method to deals with the bias issue for HELP and significantly improves the quality of bistring. Chapter 6 proposes a delay model for the HELP, which indicates the complexity of the HELP PUF for model-building attacks. Chapter 7 discusses the future work and Chapter 8 concludes the dissertation.

CHAPTER 2
A Non-Volatile Memory Based Physically Unclonable Function without Helper Data

2.1 Introduction

Many hardware security and trust mechanisms depends on a secret key which serves as a unique, un-reproducible ID for each integrated circuits (IC). Conventional methods of storing secret keys involves programming the embedded secrets in ROMs, e-fuses or non-volatile memories. Such non-volatile keys unfortunately are vulnerable to invasive physical attacks by which the secret keys may be extracted from the storage device. Once a secret key is compromised, adversaries may make clones of the IC that own the same key, thus defeating the security mechanism.

Physically Unclonable Functions (PUFs) are emerging as an alternative to conventional approaches to storing secret keys in ICs. PUFs extract entropy from variations in the physical and electrical properties of ICs, that are unique to each IC, as a means of generating secrets. Secret keys derived from PUFs mitigate the vulnerabilities of embedded digital keys mainly in two ways: (1) the generated secret keys are volatile and only present in a digital form when the chip is powered on and running. thus making it difficult for adversaries to steal by invasive attacks; (2) even if the secret key is known, it is intractable for the manufacturer to duplicate a second chip with an identical key. These secrets can be used in various security applications including device identification, authentication, metering, remote activation and encryption [1].

Applications such as encryption require precise regeneration of the secret bitstring, possible under different environmental conditions. This requirement presents challenges for PUFs (in contrast to secrets that are programmed into non-volatile memories or NVMs) because the entropy source leveraged by PUFs is analog in nature and hence can be significantly impacted by changes in temperature and voltage (TV) noise. Moreover, distinguishing subtle differences in the entropy source is further challenged by measurement noise in many cases. Typically, a secret bit is generated according to the entropy difference extracted from two identical structures, in which way an approximately equal number of “1”s and “0”s can be generated depending on process variations. For instance, if the entropy source of the former structure is larger than the latter, a “1” is generated; otherwise a “0”. Consider a case where the difference of the two compared entropy is so tiny that they will fluctuate across each other under environmental noise (or measurement noise), a bit flip will occur. Therefore, secret bitstrings generated by PUFs can not be directly applied into cryptographic applications as secret keys, until the error bits get corrected or eliminated.

The most popular approach to dealing with these challenges is to extract error correcting information from the secret bitstring once it is generated for the first time during the enrollment process, that is later used to correct errors which occur during regeneration [2]-[6]. The error correction information is stored in reliable, digital storage, e.g., in an on-chip NVM or an off-chip storage device. Generally, two drawbacks come along with storing such correction information: (1) implementation as well as storage for correction information inevitably incur hardware overheads to the overall design; (2) the correction information reveals part of the secret information of the generated keys, rendering adversaries easier to emulate PUF behavior through model-building attacks [4]. A second thresholding-based technique 'avoids' bit flips by being selective regarding which components of the entropy source can be compared reliably to generate a bit [18]. However, thresholding techniques also require helper data that indicate which comparisons are reliable.

In this chapter, we propose a NVM-based PUF implementation that does not require helper data for regeneration. The entropy leveraged in our scheme is the manufacturing variations that occur in the transconductanc1e (or resistive) characteristics of the NVM cells. The enrollment process measures and digitizes these variations and then 'programs' the NVM cells with the random bitstring that is produced. Therefore, the full reliability of the NVM is used to preserve the bitstring across power cycles and under varying TV conditions, which allows regeneration processes to extract it without suffering bit-flip errors.

The enrollment process is carried out in a special manner. First, elements of the PUF's entropy source are stimulated and digitized using an on-chip measurement structure that is capable of providing 'soft information'. Soft information implies that the magnitude of the analog variations are digitized into multi-bit, e.g., 8-bit, digital values. A distribution is then constructed using these digital values and a median-finding algorithm is used to partition the population into two segements (with an equal number of elements in each segment). NVM cells with digital values in the lower half of the distribution are programmed with a '0' while those in the upper portion are programmed with a '1'.

Our proposed NVM PUF, by its very nature, defeats one of the stated advantages of PUFs, i.e., PUFs eliminate the cost of including NVM on the chip and the need to store the secret bitstring in digital form. However, the NVM PUF does preserve the basic premise of a PUF, namely, that the secret is derived from manufacturing variations and is not programmed (or even known) by the manufacturer as is true in the traditional use of NVM. The real benefit of our proposed scheme is in the use of NVM cells as both a source of entropy and a means of eliminating public 'helper data'. We recognize that storing the secret in NVM memory represents a vulnerability and may disqualify the NVM PUF for high security applications that need to protect against invasive probing attacks. However, the small footprint and the guarantee of high reliability of the NVM scheme make it attractive for other, lower security, small form factor, applications.

In this chapter, we describe the NVM PUF enrollment and regeneration processes in the context of Memristor devices. Published data on within-die variations in Memristor arrays is leveraged to show proof-of-concept and to guide the design of on-chip measurement infrastructures which are capable of measuring and digitizing NVM cell resistance variations.

We introduce the background information in Chapter 2. The scheme overview of the proposed PUF design as well as the basics of Memristor devices are introduced in Chapter 3. The proposed voltage-to-digital converter and the PUF architecture are presented in Chapter 5, and Chapter 6 evaluates the proposed strategy using measured data from published literature. Chapter 7 concludes the chapter.

2.2 Background and Related Work

This section introduces some background information that are related to the proposed non-volatile memory based PUFs. The first section introduces several important metrics that are generally used to measure the quality of a PUF, in which the reliability metric is what this proposed research focuses on. This is followed by section 2.2 that summarizes two classes of existing schemes that have been proposed to address the reliability issue for PUFs. Since this research will describe the proposed scheme in the context of Memristors, section 2.3 gives a brief survey of several recently proposed PUF designs that are based on Memristor devices.

Various types of PUFs have been proposed depending on different electrical or physical characteristics they exploited as entropy sources, e.g., delay-based PUFs, resistance-based PUFs, threshold-voltage based PUFs and so forth. One of the important classes is memory-based PUFs, where SRAMs are investigated as conventional PUF cells and its variations on the bi-stable characteristic are exploited as entropy source. Recently, emerging memory techniques have been proposed as alternatives to conventional non-volatile memories (like flash) and they have attracted enormous amount of attention from researchers in the computing world due to their advantages. Compared to conventional memory technologies, emerging memory technologies have faster speed, better density and lower power consumption. Proposed emerging memory technologies includes Resistive random access memories (ReRAMs), Phase Change Memory (PCM), Spintronics-based technology like Spin-transfer torque RAMs (STT-RAMs) and Spin-Orbit torque RAMs (SOT-RAMs). Correspondingly, a few implementations that use these emerging technologies as PUF designs have recently been investigated.

The feasibility of building a Memristor-based PUF is recently discussed in [20]. The authors utilized a weak write mechanism which leverages the resulting unpredictable logic states to implement a PUF. Evaluations in the paper only focused on the uniqueness metric without consideration for the stability of the PUF. The work in [21] proposed a possible memristive PUF configuration that is based on the randomness of the resulting programming state of two cells in series that occurs after a reset operation. Then a Public PUF (PPUF) protocol that is implemented by memristors is also discussed. Garrett S. Rose et al. proposed a write-time based memristor PUF that leverages the write time variability of the Memristor device [22]-[23]. It is implemented by choosing the actual SET time close to the minimum SET time so that the percentage of the output logic '0' or logic '1' will be each nearly 50%. The “minimum set time” is determined by using a Monte Carlo simulations on a variable mobility memristor model. Another PUF that integrates a Memristor device into the conventional RO-PUF structure is proposed in [24]. Variations in high state and low state resistance after a programming operation is used as the entropy source. The authors demonstrate that the randomness in the resistance values increases the number of CRPs of conventional RO-PUFs.

Authors in [25] proposed a PUF based on Spin-Transfer Torque (STT) Magnetic RAM, responses are generated by comparing the inherent random mismatches between the resistance of STT_MRAM cells of the same state. An Automatic Write-Back (AWB) technique is proposed is enhance the reliability of the proposed PUF designs. Le Zhang et al. proposed a Phase Change Memory based PUF that exploits process variations and programming sensitivity of the device [26]. The paper proposed two prototypes for implementing a PUF design where the first is to exploit randomly varying pulses, and the second is to count the number of fixed programming pulses required for a target value. Another STT MRAM-based PUF is proposed in [27], in which the standard 1T1MTJ STT memory cell is used and a differential read scheme is used to generate PUF responses. Major metrics of the PUF are evaluated using SPICE simulations where device models are extracted from open literatures. Compared to [27], authors in [28] proposed a geometry based Magnetoresistive RAM (MRAM) PUF and verified major metrics using both simulation and fabrication results. A authentication protocol that is based on the proposed MRAM-PUF as well as a systemic MRAM PUF architecture are also proposed. The paper also evaluated the hardware overheads for the authentication algorithm implemented by the proposed PUF system in terms of area, power and delay overheads.

In contrast to previous work, the primary goal of our proposed PUF is to eliminate bit flips and the need for any type of helper data. To realize this goal, we exploit the characteristic that the bi-model resistance profiles of Memristor devices are widely separated and therefore, the membership of a specific device in either profile can be determined reliably. Other contributions of this paper include 1) a stimulus circuit and an on-chip voltage-to-digital converter (VDC) scheme for obtaining soft information on the resistance characteristics of the NVM cells and 2) a median-finding algorithm that is robust to non-Gaussian resistance distribution profiles.

2.3 Scheme Overview and Memristor Devices Basics

This section first presents an overview of the proposed PUF scheme and then discusses some basics of a Memristor device.

2.3.1 Overview of the Proposed Scheme

As indicated in section 2.1, the elimination of helper data is a major benefit of the proposed NVM PUF. Fig. 2.1 illustrates the mechanism by which this accomplished. The resistance distribution shown along the bottom left illustrates the randomness that exists in the resistance of the Memristor cells programmed in the low resistance state (LRS). This distribution represents the entropy source for the NVM PUF. As described in the next section, the analog resistance values are digitized and the median of this distribution is determined. The histogram shown in the upper left depict the profiles after a selected

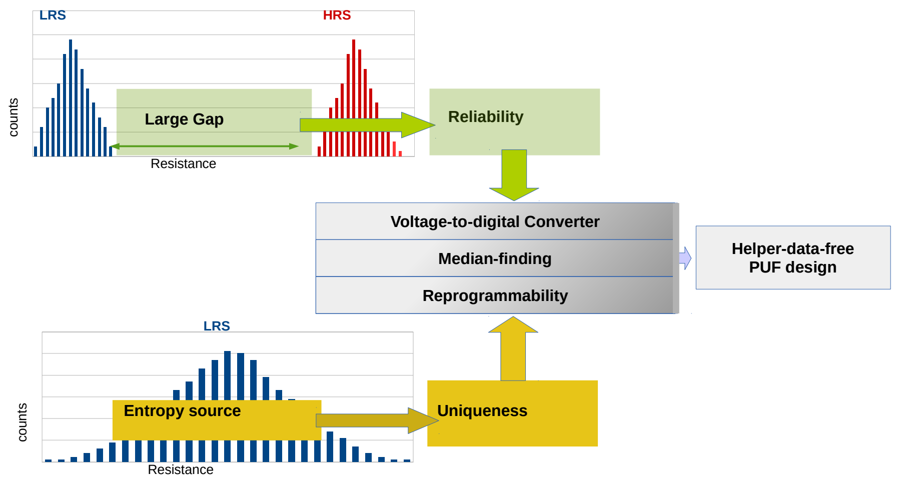


Fig. 2.1 Overall concept of the NVM PUF design

set of NVM cells are reprogrammed into the high resistance state (HRS). The large gap between these profiles ensures that subsequent regenerations always make the correct decision regarding the profile to which a given NVM cell belongs. This is a key distinguishing feature of the NVM PUF. All other PUFs (to our knowledge) must operate on the LRS distribution for regeneration. Unfortunately, varying TV conditions and measurement noise change the LRS distribution profile, making it impossible to generate the same bitstring without helper data.

Fig. 2.2 further illustrates the enrollment process in more detail. The flow starts with programming all the memristor cells to the low resistance state, by using the “write-operation” of memristors which is described in the next section. A voltage-to-digital converter (VDC) is then responsible for digitizing the sensed voltage drop across each memristor cell to a digital value between 0 to 128. All these digitized values are stored in an SRAM array that is cell-to-cell mapped to the memristor arrays. In order to create a histogram, a state machine is used to count the number of instances for each digitized value (from 0 to 128). This is achieved by storing the counts of each digital value into a second 129-cell on-chip SRAM, whose addresses represent the counters storing the corresponding digital values. Then a state machine is utilized to find the median digital value of the profile by adding up the counter values from low to high addresses. The median value is then recorded and used as a divider that determines which memristor cells are going to be reprogrammed to the high resistance state (those with larger values than the median). All cells will be randomly split into two equal-numbered groups, LRS and HRS profiles, after the reprogramming procedure. The random variations of LRS determine how the resistance profiles for an specific NVM array

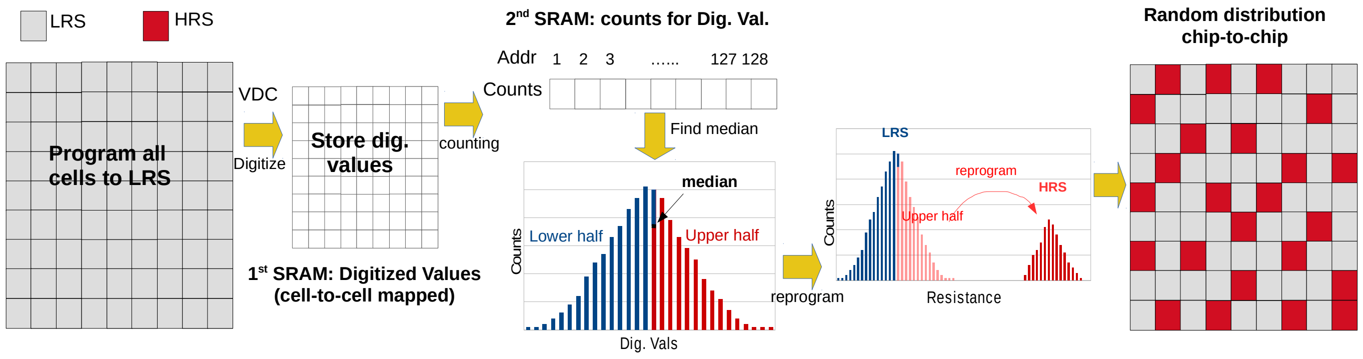


Fig. 2.2 Overview of the enrollment strategy for the proposed NVM-based PUF

distributes, with each generating a unique pattern as shown on the far right in Fig. 2.2 . Therefore the generated bitstring between two different chips will be unique.

2.3.2 Memristor Basics and Key Features

Memristors have become a mainstream research topic because of their advantages as novel memory primitives over conventional memory technologies including static RAMs (SRAMs) and Flash memories. For example, Memristors have intrinsically higher density, faster access speed and better energy efficiency [29]. Memristors are also classified as a NVM technology, in which special write operations can be used to configure cells into one of two (or more) resistance states.

A Memristor is an electrical switch that is able to retain internal resistance states according to its history of applied voltage and current [30]. The different resistance states can be sensed to generate logic '0's and '1's. Memristor write and read operations are implemented by applying write or read voltage pulse patterns. Different patterns are used for the reading and writing operations.

Fig. 2.3 shows the structure of a Memristor cell and the mechanisms used for read and write operations. As shown in Fig. 2.3 (a), a Memristor cell is composed of two electrodes and a metal oxide doping layer sandwiched between them. The length of the doping region *w* will be extended to the maximum length of D when the dopants are fully constructed (doped), and reduced to 0 when dopants are completely destroyed (undoped). The resistances of the completely doped region and undoped region can be represented by R_{on} and R_{off} respectively. Equation (1) gives an expression for the overall resistance as a function of the doping extent *w*.

$$R(w) = R_{on} \cdot \left(\frac{w}{D}\right) + R_{off} \cdot \left(1 - \frac{w}{D}\right) \tag{1}$$

The doping behavior can be controlled by applying voltage pulses of the appropriate magnitude and duration as shown in Fig. 2.3 (b). The change in the doping characteristic of the Memristor cell changes its resistance characteristics. This is depicted in the figure and labeled as LRS for low resistance state and HRS for a high resistance state, which corresponds to a logic '1' and '0', respectively. To write a logic '1', V_{in} should generate a positive square voltage pattern with magnitude V_A and time duration T_{W1}. To ensure a successful write, the magnitude of V_A must be larger than the threshold write voltage of $V_{th,w1}$ and the duration T_{W1} must be longer than $T_{th, w1}$. Similarly, the operation for writing a logic '0' requires a negative write voltage $-V_A$ with duration of at least $T_{th, w0}$. Note that some memristors cannot be configured properly after manufacture until being conditioned with a larger formation voltage V_f [32]. We assume memristors used in our enrollment algorithm have been “formed” for normal configurations. To perform a read operation, a voltage pulse pattern is required that is composed of a negative pulse followed by a positive pulse with equal magnitude and duration [31]. The negative pulse is used to detect the current internal state but it also perturbs the doping state of the cell. The subsequent positive pulse is designed to re-generate the doping conditions and corresponding resistivity of the original state. This pattern of read pulses is illustrated in Fig. 2.3 (c), which also shows when the corresponding output value is available for reading, in particular, the intervals t_1-t_2 for read '1' and t_4-t_5 for read '0'.

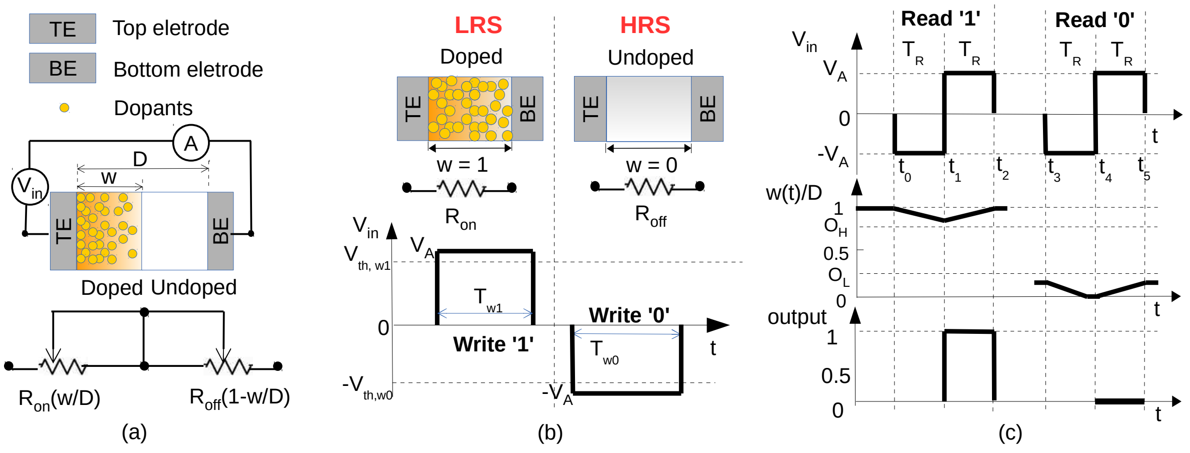


Fig. 2.3 The structure of a memristor cell and its write and read scheme. (a) memristor device structure and equivalent model [30]. (b) Write scheme. (c) Read scheme [31].

Fig. 2.4 shows the histogram of the HRS and LRS variations extracted from a 1600 Memristor devices (40*40 nanocrossbar array) [33]. The spread in the distributions illustrate that the resistance of a Memristor cell after a write operation varies considerably, and is due to process variations and voltage variations over the Δt of the write operation. This characteristic makes it challenging to use Memristor cells for a PUF in cases where the resistance variations within either of the two states are used as the source of entropy. This is true since the read operations that take place during regeneration also change the resistance characteristics of the cells, which in turn increases the probability of a bit flip.

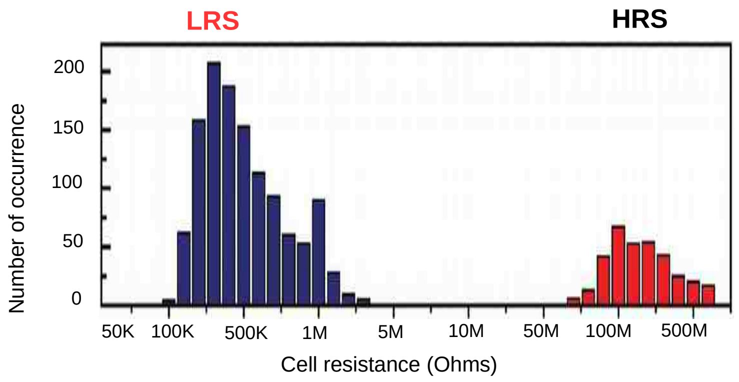


Fig. 2.4 Histogram of the HRS and LRS resistances variations extracted from a 40*40 nano-crossbar array (1600 devices) [33].

2.4 VDC and Proposed PUF Architectures

In this section, a voltage-to-digital converter is introduced that is used to digitize the sensed analog voltage from the Memristor cells. This is followed by presenting a circuit level modification that is able to use the Memrisor cells as PUF primitives. The enrollment scheme that is based on these two critical modules is illustrated in detail to demonstrate how we can use a median-finding algorithm to eliminate helper data.

2.4.1 Voltage-to-Digital Converter (VDC)

The PUF structures that we propose require the measurement and digitization of a value proportional to the resistance of Memristor cells in the LRS. The proposed architectures, which are described in the following sections, provide a voltage from a voltage divider network(s) that is proportional to the LRS resistance.

The voltage-to-digital converter (VDC) shown in Fig. 2.5 is capable of digitizing these voltages [34]. The VDC has two voltage inputs, labeled VoltInUpper and VoltInLower, two digital inputs labeled e_1 and e_2, and two delay chains (upper and lower) connected to a sets of latches. The voltage inputs connect to NFET transistors inserted in series with the odd-numbered inverters of the delay chains. Voltages less than V_{DD} introduce additional delay through these inverters that is proportional to the applied voltage as an edge propagates down the inverter chains.

The function of the VDC is to create an 8-bit digital value between 0 and 128 that is related to the voltage present on the VoltInLower input. This voltage is derived from the voltage divider network and is always smaller than the supply voltage (V_{DD}). The digitization process is started by the Edge Generator, which launches a rising edge onto e1 and then after some delay, a second rising edge onto e2 as shown in the figure. Under the condition that the voltage on VoltInUpper is sufficiently larger than the voltage on VoltInLower, the e2 edge catches up and passes the e1 edge. The latches on the outputs of the even inverters in the delay chains record the point at which this happens as a thermometer code (TC). A TC is a sequence of '0's (or '1's) followed by a sequence of '1's (or '0's). The number of '1's (or '0's) in the TC reflects the magnitude of the difference between the two applied voltages. We refer to the number of '1's in the 128 latches connected to the lower chain as a TCV. In our proposed implementation, the voltage applied to VoltInUpper is V_{DD} as a means of ensuring that it is always larger than the voltage to be digitized on VoltInLower. The wide range of resistance variations that occur in the LRS states of Memristors cells produces a wide range of voltages that need to be digitized by the VDC. Moreover, TV environmental variations also impact the timing behavior of the VDC. The Edge Generator component of the VDC is used in a calibration process to ensure that the VDC is able to produce useful digital values under these

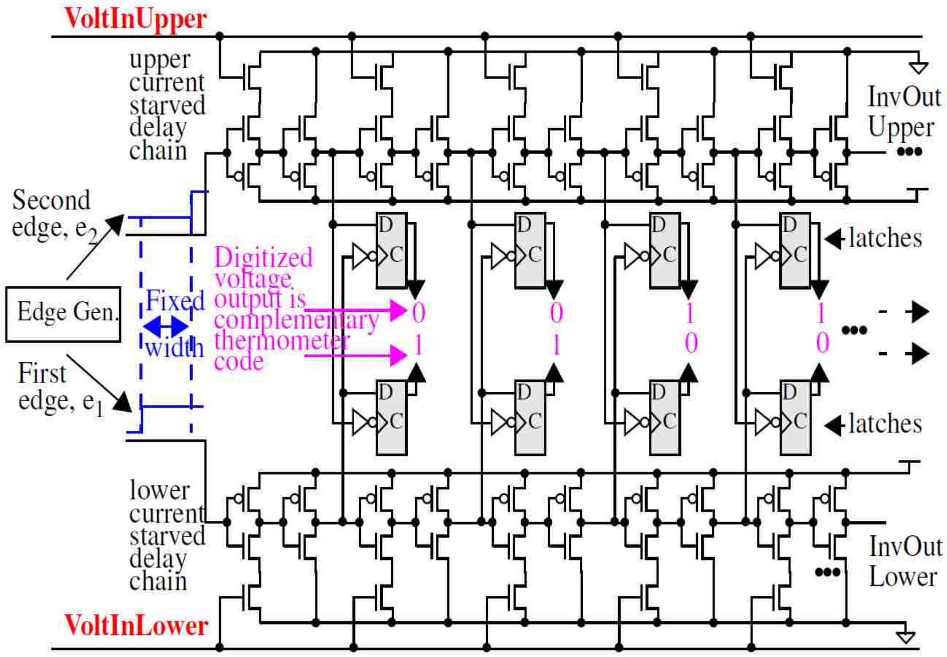

VoltInUpper
upper current starved delay chain
InvOut Upper
Second edge, e2
Edge Gen.
Fixed width
First edge, e1
Digitized voltage output is complementary thermometer code
latches
latches
lower current starved delay chain
InvOut Lower
VoltInLower

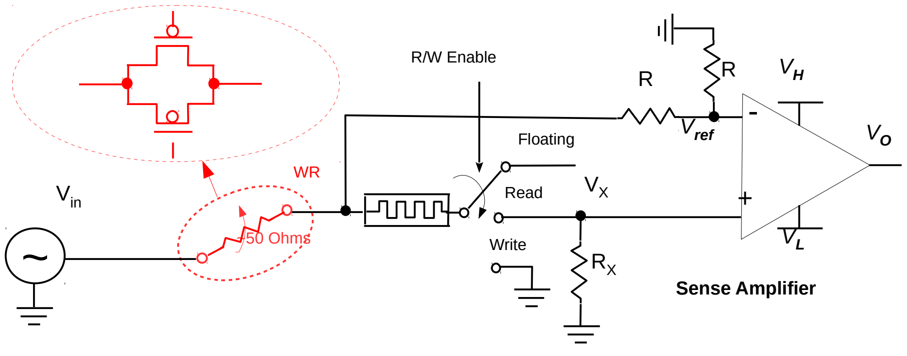
Fig. 2.5 Voltage-to-Digital Converter (VDC)

conditions, where 'useful' is defined as values above 0 and less than the overflow value of 128. Calibration tunes the Δt between e1 and e2 edges, maximizing the sensitivity of the VDC to specific ranges of voltages, and allowing it to accommodate for TV variations. The transfer curve characteristics and calibration process are described below in the context of an example.

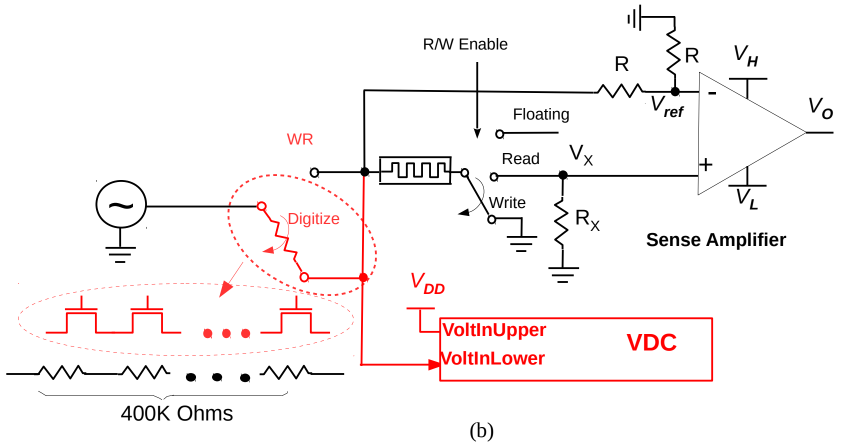
2.4.2 Proposed Memristor PUF

Fig. 2.6 (a) shows the architecture proposed in [31] for a Memristor-based NVM, with the exception of the switch on the left side of the diagram (which is needed only for the PUF). The resistance of this switch, implemented as a pass gate with very wide transistors, is very low, e.g., on order of 50 Ohms or less.

This switch is closed when the memory is accessed for normal read and write operations. In this case, the Pulse Generator labeled V_{in} delivers pulses to a selected set (or word) of Memristor cells according to the diagrams shown earlier in Fig. 2.3 . For normal read operations, the R/W Enable switch is set to the 'Read' position, which creates a voltage divider network between V_{in}, across the Memristor cell and resistor R_x to ground. The resistance of R_x is set to a value of approx. $(R_{off} + R_{on})/2$ so that V_X will be larger than V_{ref} (half of V_{in}) when the cell is programmed to its LRS and smaller than V_{ref} when programmed to HRS. In this way, V_O will be V_H (logic '1') when the Memristor cell is in LRS and V_L (logic '0') when the cell is in HRS. From the distributions shown in Fig. 2.4 , the value of Rx would be approx. 10 MΩ. The modifications shown in red in Fig. 2.6 (b) are required in order to allow the Memristor memory to be used as a PUF. The large, low resistance switch is disabled and instead a high resistance, approx. 400 KΩ, switch is enabled. This switch is also connected in series between the Pulse Generator and the Memristor array. The value of 400 KΩ is the resistance near the midpoint of the distribution of LRS programmed Memristor cells from Fig. 2.4 . Therefore, when a Memristor cell that is programmed in its low resistance state is enabled, the voltage on the voltage divider network is a value between 200 mV and 882 mV (with V_{DD} at 1.0V). These values are obtained by using the extreme values of the LRS distribution in Fig. 2.4 . For example, 200 mV is obtained from the voltage divider network expression (100 KΩ/500 KΩ). This voltage is delivered to the VoltInLower input of the VDC, as shown



(a)



(b)

Fig. 2.6 (a) Circuit structure proposed Memristor memory [31]; (b) Modifications needed for proposed Memristor PUF.

along the bottom of Fig. 2.6 (b).

Most memory architectures are byte or word addressable, which means that multiple Memristor cells are accessed simultaneously. The arrangement shown in Fig. 2.6 (b), on the other hand, assumes that each Memristor cell is individually addressable, i.e., the word-size of the PUF implementation is 1 bit. Therefore, an architecture level change is needed in addition to the components of Fig. 2.6 (a) and Fig. 2.6 (b) in order to convert the Memristor array into a PUF.

2.4.3 Enrollment Algorithm

As indicated earlier, the enrollment process leverages the random resistance variations in the Memristor cells as the source of entropy, and then uses the programmability of the Memristor cells to eliminate helper data. The enrollment algorithm that accomplishes these goals is given as follows:

1. The controller for the memory is instructed to program all Memristor cells to the low resistance state. This is accomplished as a 'normal' write 1 operation as described earlier with the large, low resistance pass gate switch enabled.

2. The controller is again instructed to sequence through a set of write operations but this time with the high resistance switch enabled and exactly one Memristor cell selected, i.e., the R/W enable signal is set to 'Write' while all other cells in the array are set to 'Floating'.

3. Immediately after the write pulse is asserted, a start signal is issued to the VDC to begin the digitization process.

4. The 8-bit digitized value from the VDC is stored in an on-chip SRAM memory at

the address corresponding to the tested Memristor cell.

5. Once all cells in the Memristor array are digitized, a state machine creates a histogram of the digitized voltages stored in the SRAM. The histogram is created by using the digitized values as an address into a second on-chip SRAM, whose storage locations represent counters recording the number of instances of a particular digitized voltage.

6. A state machine parses the histogram data from low to high address, adding up the counter values. The memory address of the median value, which partitions the array of elements into two equal-sized groups, is recorded.

7. The state machine then parses the first SRAM, comparing the stored digitized voltage with the median. The Memristor array is again placed in normal write mode and those cells whose value exceeds the median are re-programmed to the HRS.

8. A bistring is constructed using a sequence of normal read operations, which are designed to preserve the LRS or HRS of the programmed Memristor cells. The sequence of read addresses can be generated as a linear sequence or by using a linear-feedback-shift-register to generate the sequence pseudo-randomly.

The ordering of the Memristor cells from left to right within the histogram is random for each chip, and therefore, the bitstrings will be unique across chips. Also, the large threshold between the two distributions makes it possible for the bit generation algorithm to succeed in reliably making the same decision about whether the Memristor cell is in a LRS or HRS, thereby eliminating the need for helper data.

Note that resistor divider network reduces the 'write' voltage during the digitize operation, in most cases to a value below the threshold shown in Fig. 2.3 (b). Therefore, changes in the actual resistance value are likely to be very small. However, the enrollment process as described is robust to these types of resistance changes so they are of no consequence.

2.5 Evaluation Using Measured data

This chapter demonstrates the practicability of the enrollment process using the measured data from [33]. Fig. 2.7 shows that resistance variations of the LRS programmed Memristor devices ranging from approx. 100 KΩ to 3 MΩ, and the profile does not have to be Gaussian. A robust feature of our proposed median finding algorithm is that we do not need to build the voltage divider network and VDC to digitize this entire range. In fact, only the values in the middle of the distribution, i.e., in the range of 200 KΩ to 1 MΩ, need to produce non-underflow (0) and non-overflow (128) TCVs within the VDC.

The transfer curves in Fig. 2.8 indicate that the VDC operates best for VoltInLower values in the range of 300 mV to 800 mV (for V_{DD} of 1V), where it produces TCVs in the range from 5 to 120. Note that this range can be adjusted using calibration to accommodate process and TV variations, as shown by the dotted curves. Calibration tunes the Δt between e1 and e2, effectively shifting the curves horizontally. Setting the high resistance switch in Fig. 2.6 (b) to approx. 400 KΩ produces voltages of 333 mV when the

Memristor cell is 200 KΩ and 714 mV for Memristor cells at 1 MΩ. Such calibrated voltage range (from 333 mV to 714 mV) fits the best region for voltage digitization of VDC.

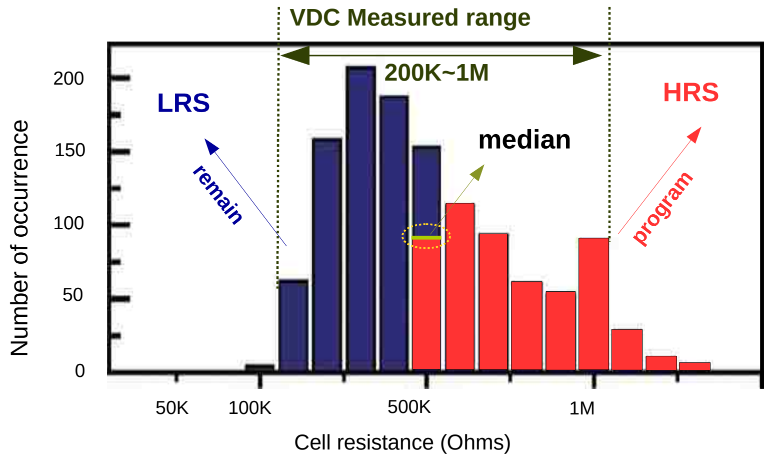


Fig. 2.7 VDC measured range for the measured LRS data profile from 1220 memristor cells in [33].

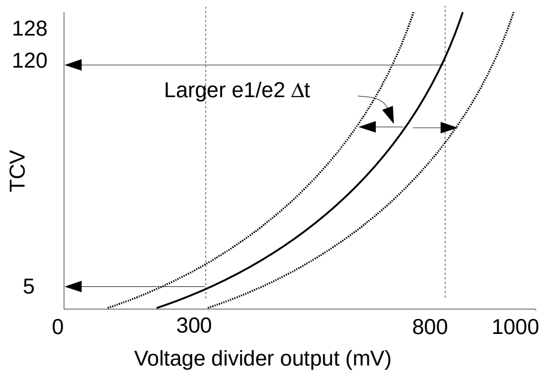


Fig. 2.8 Typical transfer Curves for VDC.

CHAPTER 3

PUF-Based Authentication

In the context of hardware systems, authentication refers to the process of confirming the identity and authenticity of chip, board and system components such as RFID tags, smart cards and remote sensors. The ability of physical unclonable functions (PUF) to provide bitstrings unique to each component can be leveraged as an authentication mechanism to detect tamper, impersonation and substitution of such components. However, authentication requires a strong PUF, i.e., one capable of producing a large, unique set of bits per device, and, unlike secret key generation for encryption, has additional challenges that relate to machine learning attacks, protocol attacks and constraints on device resources. In this chapter, we describe the requirements for PUF-based authentication, and present a PUF primitive and protocol designed for authentication in resource constrained devices. Our experimental results are derived from a 28 nm Xilinx FPGA.

3.1 Introduction

Authentication is traditionally characterized as a process that verifies “something you know”, e.g., a password, “something you have”, e.g., hardware one-time-password tokens, and “something you are”, e.g., your fingerprints. Multi-factor authentication requires two or more of these components from different categories. PUF-based authentication provides individual devices with a set of passwords (bitstring responses to challenges) that uniquely identify it (a fingerprint), so in this sense, it can be characterized as a multi-factor authentication mechanism. PUFs derive their fingerprint from random variations that occur in the manufacturing process of a chip or board. For example, a delay-based PUF measures and digitizes variations that occur in paths and/or gates within the chip or along wires in a printed circuit board (PCB) [35]. Although we use variations in path delays as the entropy source in this paper, there are many other sources of variations that can be leveraged, as is evident from the published literature on PUFs.

PUFs have been proposed for other types of applications including encryption, for detecting malicious alterations of design components and for activating vendor specific features on chips. Each of these applications has a unique set of requirements regarding the security properties of the PUF. For example, PUFs that produce secret keys for encryption are not subject to model building attacks (as is true for PUF-based authentication) which attempt to ‘machine learn’ individual path delays for a chip as a means of predicting the complete response space of the PUF. This is true for encryption because the responses to challenges are typically not ‘readable’ from an interface on the chip. In general, the more access a given application provides to the PUF externally, the more resilience it needs to have to adversarial attack mechanisms.

Authentication as an application for PUFs clearly falls in the category of extended access. The term ‘hardware token’ or *prover* is typically used to identify a fielded device that embeds the PUF, such as a smart card, and the term ‘secure server’ is used in reference to the verifier.

Applications such as authentication require a **strong PUF**, i.e., a PUF that can produce a very large number of challenge-response-pairs or CRPs. Challenges and responses are the digital inputs and corresponding outputs of the PUF. In order for authentication to work, it must be necessary and impractical for an adversary to apply all possible challenges to the PUF on a chip as a means of obtaining all of its responses. Making this infeasible makes it impossible for an adversary to build a ‘clone’ of the chip that replicates the CRP behavior. However, the requirement of a very large CRP space is, in general, challenging to meet for PUFs. It requires a large source of entropy, which can become expensive area-wise when the PUF is implemented using dedicated and specialized components.

Authentication is typically characterized as having two phases: enrollment and regeneration. Enrollment occurs immediately after manufacture and involves the verifier generating a random set of challenges which are applied to the token to generate a corresponding set of responses. The set of CRPs are stored on the verifier for each chip individually in a secure environment. The stored CRPs can then be used to carry out authentication in the field with the token. The verifier only needs to store a limited set of CRPs in the secure database because the very large CRP space of the strong PUF combined with the secrecy of the chosen CRPs makes it difficult or impossible for an adversary to know how to respond using a clone of the token.

Bear in mind, authentication can also be implemented by having the PUF generate a secret key for encrypting communication between the prover and verifier. The enrollment process involves the PUF generating a shared key that is stored on the server through a one-time interface, i.e., an interface that can be disabled, along with helper data. The helper data is later transmitted to the token as needed for authentication in the field to enable precise regeneration of the key. The token in this scenario needs to incorporate an encryption algorithm, which adds to the required resources. Although this method requires only a weak PUF that is capable of producing only a small number of bits (a plus), the encryption operation carried out by the token is subject to side-channel attacks that attempt to learn the key (a minus). Once learned, the security mechanism is defeated. Therefore, strong PUFs that have a very large CRP space provide an advantage by making it infeasible for an adversary to extract all the secrets embedded in each token.

Most authentication proposals also limit the amount (or eliminate completely) the need for helper data and instead allow for fuzzy matching to occur between server stored responses and those generated in the field by the token. In other words, a *small* number of differences are tolerated in the response bitstrings. Although fuzzy matching reduces the storage requirements for the verifier by eliminating the helper data, it also increases the possibility of aliasing and impersonation, i.e., the likelihood that two devices produce the same responses (within the noise margin).

In this chapter, we propose a hardware-embedded delay PUF called HELP as a strong PUF for authentication. HELP leverages entropy present in functional units already present in the chip, and therefore, it does not require the insertion of dedicated components. Moreover, the overhead associated with integrating HELP into functional unit is very small relative to the size of the functional unit. HELP is unique in that it leverages delay variations in structures that are not identical and implicitly provides tamper protection of the functional unit(s). This paper contributes beyond previously published work in [36]Error: Reference source not found in the following ways:

- We implement HELP on a Xilinx 28 nm 7020 Zynq chip embedded on AVNET's Zedboard [38] using both glitchy and glitch-free functional units as the source of entropy and analyze the statistical quality of the bitstrings.
- We isolate and analyze entropy introduced from multiple sources and discuss the trade-offs and impact on security.
- We propose an authentication protocol using HELP.

3.2 Related Work

An excellent survey and critical review has been recently published that covers the state-of-the-art with regard to PUF authentication for resource constrained devices [39]. The criteria used to review the existing methods assume a low-cost resource constrained token and resource-rich server, and the use of a strong PUF. The authors indicate that protocols which require NVM are less attractive because of the increased cost of manufacturing of NVM components in CMOS technologies and because of recently disclosed vulnerabilities of NVMs to probing attacks. The PUF protocols proposed in [40]-[56] are evaluated against the following characteristics [39]:

- Resilience to measurement and temperature/voltage (TV) noise sources.
- Resilience to machine learning via use of cryptographic hash functions and XOR functions as needed.
- Are techniques needed to expand the response space (PRNG) of the strong PUF?
- Ease of instantiation of the PUF authentication mechanism.

- Resistance to protocol attacks, i.e., token and/or server impersonation and denial of service attacks.

The authors conclude that the main problems with the protocols are rooted in the PUF itself and that research should focus on developing a truly strong PUF with solid cryptographic properties.

3.3 Overview

3.3.1Goals and Objectives

One of the goals of this work is to isolate and characterize the main sources of delay variations (the entropy source) on the chip, namely, 1) within-die delay variations that occur within individual FPGA LUT primitives, 2) global delay variations that occur across all LUTs on the chip and 3) delay variations introduced by static and dynamic logic hazards. All of these sources of variations change the delay characteristics of paths uniquely on each chip.

A key objective is to determine the magnitude of these variations with respect to measurement and temperature/voltage (TV) noise sources. We refer to this noise as “TV noise” since TV dominates even when repeated sampling and TV compensation techniques are applied. TV noise works to impede access to the entropy provided by delay variations, and reduces the amount of usable entropy. Delay variations introduced by within-die process variations are relatively small even when measured through a single LUT. On the other hand, global variations and variations introduced by hazards are well above the TV noise margin, making them attractive as a source of entropy. However, there is a downside to leveraging these larger sources of entropy as discussed below.

We integrate HELP into a GF(4) subcomponent and a full-blown GF(256) version of the Advanced Encryption Standard (AES) SBOX functional unit [57]. The GF(4) version can be implemented using a logic depth of 1, which allows individual LUT delays to be analyzed. We implement the GF(256) in two ways referred to as: *Standard*: without any type of special logic style or constraints and *WDDL*: without glitches using wave-differential dynamic logic [58]. The Standard implementation includes all three sources of entropy. Inter-chip hamming distance (HD), Inter-chip HD and the results of NIST statistical tests are reported to understand the trade-off of the two logic styles on bitstring generation and reproduction [59][60].

A *modulus* technique is used in combination with a helper data string as a mechanism to maximize the strength of the cryptographic properties of the PUF in the proposed authentication protocol. Glitch-free logic implementations of the functional unit, such as WDDL, provide a distinct advantage in resource-constrained authentication applications by reducing bit flips while improving access to the limited, but most important source of entropy, namely that provided by within-die variations.

3.3.2 Attack Scenarios and Assumptions

Traditional "resource-constrained" applications such as RFID and smart cards utilize memory, small microcontrollers and/or ASICs for implementing functions. The attack models and assumptions that we describe in the context of FPGAs can be extended to these implementations as noted below. Although HELP is proposed as an FPGA authentication mechanism in this paper, the concept and techniques presented are also applicable to ASIC implementations [36].

Secure computing using FPGAs requires encryption of the programming bitstream. Modern FPGAs integrate encryption/decryption modules, and NVM-based key storage mechanisms, to support this requirement. Beyond protecting Intellectual Property, encryption also prevents tampering with the design. Although our technique can detect tamper within functional unit(s), we assume an attacker is not able to defeat the bitstream encryption mechanism. No security mechanism, PUF or otherwise, is secure if this requirement is not met.

We consider two attack scenarios. First, the adversary can gain (temporary) possession of the token and attempt to read out all responses or enough of them to “machine learn” the entropy source. Once known, a clone can be ‘programmed’. In general, strong PUFs can significantly impede, or make impossible, the success of this type of attack. For PUF architectures in which machine learning is effective, the proposed protocols typically incorporate obfuscation mechanisms to prevent direct control of the PUF and observation of its responses. The second attack mechanism is similar except that the adversary carries out a ‘man-in-the-middle’ attack, i.e., he or she listens to exchanges between the token and the server.

. Other types of attack scenarios can be avoided. For example, some protocols require one-time interfaces to be present during enrollment but such interfaces can be ‘undone’ using focused ion beam techniques. Still other protocols require the use of small NVMs, which add cost and weaken security because ‘read-out’ mechanisms are becoming increasingly effective. Therefore, avoiding one-time interfaces and NVM is a plus.

3.4 Experimental Setup

3.4.1 HELP Overview

HELP measures path delays using a simplified version of an embedded test structure called REBEL [36]. The simplified version eliminates the delay chain component and instead samples the path delays at the capture FF directly. Fig. 3.1 shows the test setup with the 'functional unit' or FU representing the entropy source. The inputs and outputs of the FU are connected to a set of Launch Row and Capture Row flip-flops (FFs), resp.

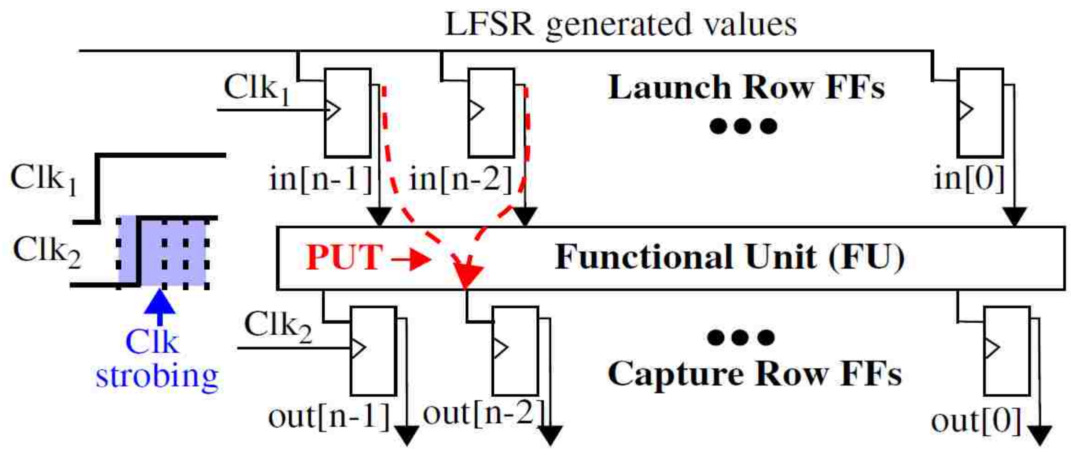


Fig. 3.1 Configuration of the AES SBOX FG(4) [57].

The delay of a path is determined using the fine phase adjust feature of a Xilinx embedded MMCM (mixed mode clock manager). A series of launch-capture clocking events are applied to the functional unit using two clocks, Clk_1 and Clk_2, as shown on the left side of . The phase shift between Clk_1 and Clk_2 is adjusted dynamically across the sequence of launch-capture tests. The digitally selected value of the fine phase shift between the two clocks is referred to as the launch-capture interval (**LCI**). The smallest

LCI interval that allows the propagating edge along a path to be captured in the capture FF is used as the digitized timing value for the path. The MMCM on the Zynq FPGA clocked at 25 MHz provides a resolution of 18 ps. Digital values between 150 (smallest LCI with value of approx. 18 ps * 150 = 2.7 ns) and 2,200 (largest LCI with value approx. 39.6 ns) are used as the path delay value. The repeated testing of the FU at different LCIs is referred to as clock strobing. The LCI used to represent the delay of a path is referred to a PUFNum or **PN**. The signed difference of two randomly selected PNs is referred to as a **PNDiff**.

3.4.2 TV Compensation

The majority of the delay variations introduced by changes in temperature and voltage is removed by applying a TV compensation process. TV compensation is carried out by computing the mean (offset) and range (multiplier) from a set of PNDiffs for each chip and for each TV corner separately. The offset and multiplier computed during enrollment are used with the offset and multiplier computed at each TV corner to compensate the PNDiffs generated at the TV corners using Eq. (3.1) And Eq. (3.2):

$$zval_i = \frac{(PNDiff_{TV_x} - \mu_{TV_x})}{rng_{TV_x}} \tag{3.1}$$

$$PNDiffs_{TVComp} = zval_i \cdot rng_{TVEroll} + \mu_{TVEnroll} \tag{3.2}$$

Here, *zval$_i$* represents a standardized PNDiff after subtracting the mean and dividing by the range computed using a set of PNDiffs produced at the TV corner, *TVx*, for a specific chip. The individual *$zval_i$* are then transformed using the mean and range computed earlier for the same chip during enrollment, i.e., at *TVEnroll*. We refer to the PNDiffs generated during enrollment as the **reference**. This linear transformation is very effective at eliminating the shifting and scaling that occurs to path delays at different TV corners (note: using the PNDiffs directly without this type of compensation does not compensate for scaling).

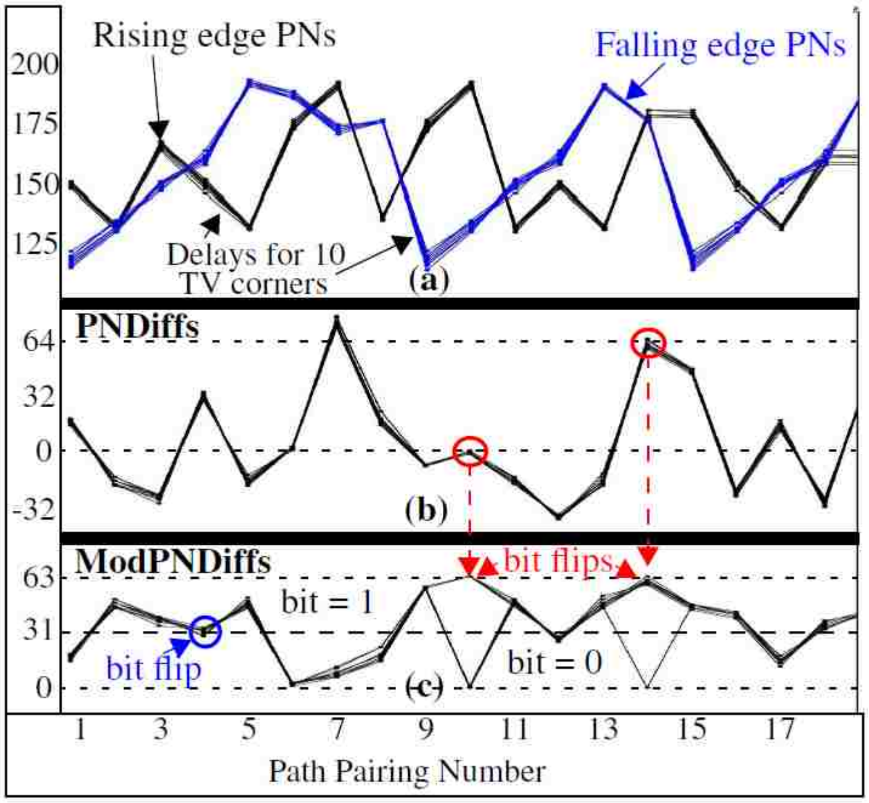


Fig. 3.2 Example rising and falling path PNs (top), random pairings of rising and fall PN differences (middle), PN differences modulo 64 (bottom).

3.4.3 Bit Generation Algorithm

The bit generation uses the signed difference in two path delays (PNDiff) as a means of both hardening the algorithm against model building and increasing the diversity in the PUF responses. A **ModPNDiff** is defined by computing a signed difference between two arbitrary selected PNs, and then applying a *modulus*. The modulus is necessary because the paths in the FU vary in length, for example, in our experiments, short paths consist of 1 LUT while the longest paths consist of 13 LUTs, which is captured in the PNDiffs. The modulus removes the ‘path length’ bias while fully preserving the smaller within-die delay variations.

For example, the top of Fig. 3.2 (a) shows two sets of waveforms labeled ‘Rising edge PNs’ (black) and ‘Falling edge PNs’ (blue). The points in the waveforms represents the delay values (PNs) measured from a set of paths in chip C_1 in the AES SBOX GF(4) experiment. Each group of waveforms with similar shape and color represent the PNs measured at each of the 10 TV corners after a TV compensation method is applied (a process identical to the TV compensation applied to the PNDiffs described above). The vertical spread in the 10 points represent uncompensated TV noise. The waveforms shown in (b) represent the PNDiffs computed from randomized pairings of rising and falling edge PNs in (a). Although only chip C_1 data is shown, the shape of the difference waveforms is similar for other chips because of the path length bias. The **ModPNDiffs** shown in (c) are the result of applying a modulus of 64 to the PNDiffs in (b). The modulus effectively ‘wraps’ all differences into the range of 0 to 63 and reduces and/or eliminates the bias. The bit generation algorithm assigns ModPNDiffs in the range from 0 to 31 as ‘0’ while those in the range of 32 to 63 are assigned ‘1’.

The red circles on points 10 and 14 show bit flips. Bit flips occur when some, but not all, of the 10 points in each group cross over one of the boundaries given by 0 or 63. An additional bit flip is shown by the blue circle for point 4, where the points cross over the boundary between ‘0’ and ‘1’. The close grouping of the 10 points makes it is possible to apply a predictive screening process that avoids most/all of these bit flips as we show below. Moreover, the modulus parameter can be used to remove bias as described but it is also useful for increasing the input-output space of the HELP PUF, which is also discussed in the following sections.

3.4.4 Functional Unit Synthesis Flow

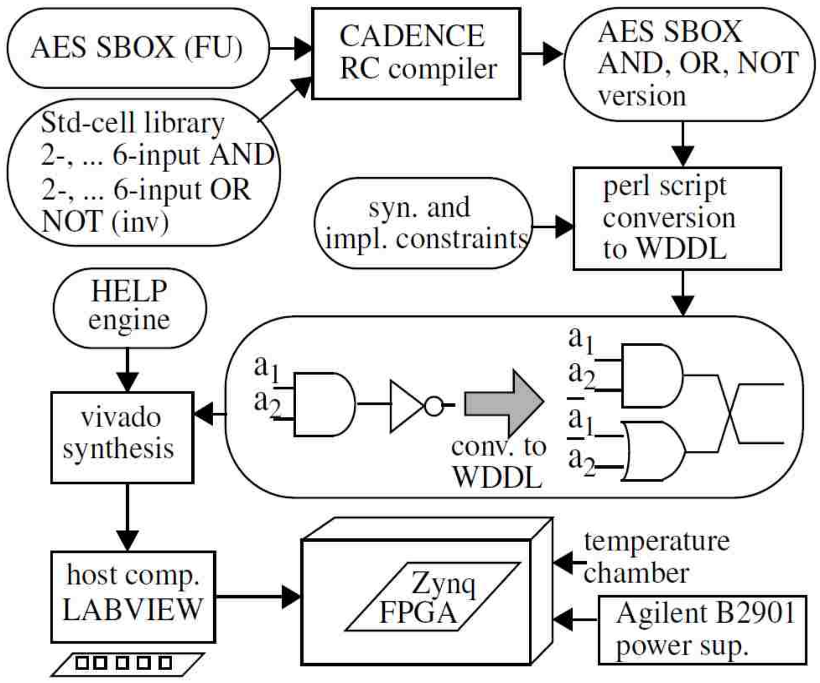

AES SBOX (FU)
CADENCE RC compiler
AES SBOX AND, OR, NOT version
Std-cell library 2-, ... 6-input AND 2-, ... 6-input OR NOT (inv)
syn. and impl. constraints
perl script conversion to WDDL
HELP engine
vivado synthesis
a_1 a_2
conv. to WDDL
a_1 a_2 $\overline{a_1}$ $\overline{a_2}$
host comp. LABVIEW
Zynq FPGA
temperature chamber
Agilent B2901 power sup.

Fig. 3.3 Process Flow

The AES SBOX is used as the functional unit in our experiments because its interconnection implementation structure is random and complex. Although only the SBOX is used in this work, the technique can be extended to the full implementation of

AES and other types of functional units (see [36] and [37]). As indicated earlier, we implement the SBOX using a special glitch-free logic style called WDDL [58] as a means of distinguishing between the underlying sources of entropy, and as a means of improving the reliability of HELP. WDDL eliminates functional and logic hazards by imposing stimulus constraints and restricting the implementation to use only AND and OR gates. WDDL is proposed as a mechanism to harden a design unit such as AES against side-channel attacks, and therefore, also attempts to eliminate information in the power curves. This latter feature is not required to improve the reliability of HELP and

therefore, we are also looking into simpler glitch-free-only strategies that have less area overhead [61]. The benefit of WDDL is that it is simple to implement and provides a nice test bed for evaluation of glitch-free logic implementation.

Fig. 3.3 illustrates the design flow followed to implement the WDDL version of the AES SBOX. A behavioral VHDL description of the SBOX along with a standard cell library are used as input to the CADENCE RC synthesis tool. The standard cell library only includes 2-input to 6-input AND and OR gates to match the LUT capabilities on the FPGA, and a NOT gate. No timing constraints were used in the synthesis and therefore, RC optimized for area.

A structural netlist consisting of only AND, OR and NOT gates represents the output of the synthesis. This file along with a set of synthesis and implementation constraints are processed by a perl script to produce a WDDL version of the netlist. One example transformation is shown in the figure where a AND gate followed by an NOT gate is converted to a complementary pair of AND/OR gates, with the outputs swapped for connections downstream as a means of emulating (and eliminating) the NOT gate.

The WDDL version therefore is constructed by creating a complementary OR gate (with complementary inputs) for all existing AND gates, and vise versa. The 8 primary inputs of the SBOX are also replicated and are driven with complementary values during evaluation. The operation of WDDL consists of two phases: a pre-charge phase in which all primary inputs (including the complementary inputs) are driven with ‘0’. This forces ‘0’s on the inputs and output of all gates throughout the circuit. The evaluate phase applies the true and complementary values to the 8 true and complementary primary inputs, resp., and causes a set of rising transitions to propagate through the circuit. For the SBOX implementation, half of the true outputs and half of the complementary outputs transition on average during evaluate. Therefore, for each of the 256 possible input transitions, i.e., from 0000000->xxxxxxxx, 8 PNs are obtained to produce a total of 2048 PNs. Another 2048 are obtained for the precharge phase, i.e., from xxxxxxxx->00000000, so a total of 4096 PNs are produced, from which a set of 2048 PNDiffs can be uniquely constructed.

From Fig. 3.3 , the WDDL version of SBOX is combined with the HELP engine (described using behavioral-level VHDL) in a project that is processed by the Xilinx Vivado synthesis and implementation tool. The constraints added by the perl script prevent the FPGA synthesis and implementation tools from optimizing the WDDL structural netlist. The programming bitstream generated by Vivado is then used to program the Xilinx 7020 Zynq chip on a Zedboard [38], which is placed in a temperature chamber.

We also synthesized AES SBOX GF(4) and GF(256) versions using a *standard* synthesis flow to serve as a comparison to the WDDL implementation. The flow for the *standard* versions simply uses VHDL descriptions of the GF(4) and GF(256) as input to the Xilinx Vivado synthesis tool without any constraints. We instantiate two copies of the GF(256) in the standard version, with the inputs to the 2nd copy complemented, to model the complementary network within the WDDL version as a means of making the two implementations as similar as possible. A similar strategy is used for the GF(4) except four copies are instantiated (each copy has only 4 inputs/outputs). The input transition sequence used for the WDDL version are also used here. Note that there are significant differences in the resource usage by the two GF(256) versions, however. For example, the standard version uses 80 LUTs in a 2-level logic structure while the WDDL version uses 756 LUTs in a multi-level logic style of up to 13 levels. The GF(4) has only 16 LUTs in 1 level of logic and therefore allows a single LUT delay to be measured.

3.5 Experimental Results

We ran our experiments on 30 copies of the Zedboard [38]. Commercial grade 7020 Zynq chips are incorporated on the Zedboard, which restricts the temperature range between 0C and 85^{oo}C and the operating voltage between 0.95 V and 1.05 V (5% around the nominal 1.00 V). The Agilent precision power supply and ESPEC temperature chamber are controlled using a LABVIEW program running on a host computer. The Zedboards were tested at 25^{o}C, 1.00 V, which we use as enrollment data, and 9 regeneration corners, which includes all combinations of three temperatures, 0^{o}C, 25^{o}C and 85ºC and three voltages, +/- 5% and nominal. The MMCM on the FPGA is configured with a 25 MHz clock frequency.

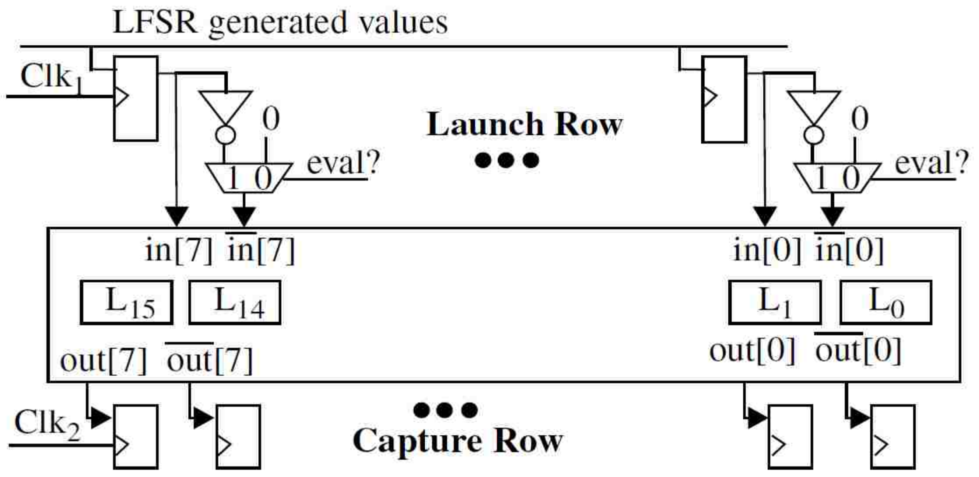


Fig. 3.4 Configuration of the AES SBOX GF(4) [57]

3.5.1 AES SBOX GF(4) Analysis

The goal of the GF(4) analysis is to determine the magnitude of within-die variations in the shortest constructible path on an FPGA, i.e., paths with 1 launch FF, 1 LUT and 1 capture FF. Fig. XXXXXXXX shows the configuration synthesized by Vivado. Two copies of the logic expressions for GF(4) given in [57], and two copies implementing their inverse, synthesized to a set of 16 4-input LUTs labeled L_{15} down to L_0. The inputs, e.g., in[7]/in[7] fan-out to the LUTs of the true and inverse copies, resp. and the outputs, e.g., out[7]/out[7], wire to a row of capture FFs. Given all inputs are applied simultaneously, there is no glitching that occurs on the outputs even though the potential exists given the diverse truth tables implemented with the LUTs. A 25 point sample of the 2048 PNDiffs measured from the 30 chips at the 10 TV corners is shown in Fig. 3.5 . The PNDiffs are computed by selecting a unique random pair (chosen by an LFSR) of PNs, one from the rising paths and one from the falling paths (see Fig. 3.2 (a)). The groups of waveforms of the same color shown along

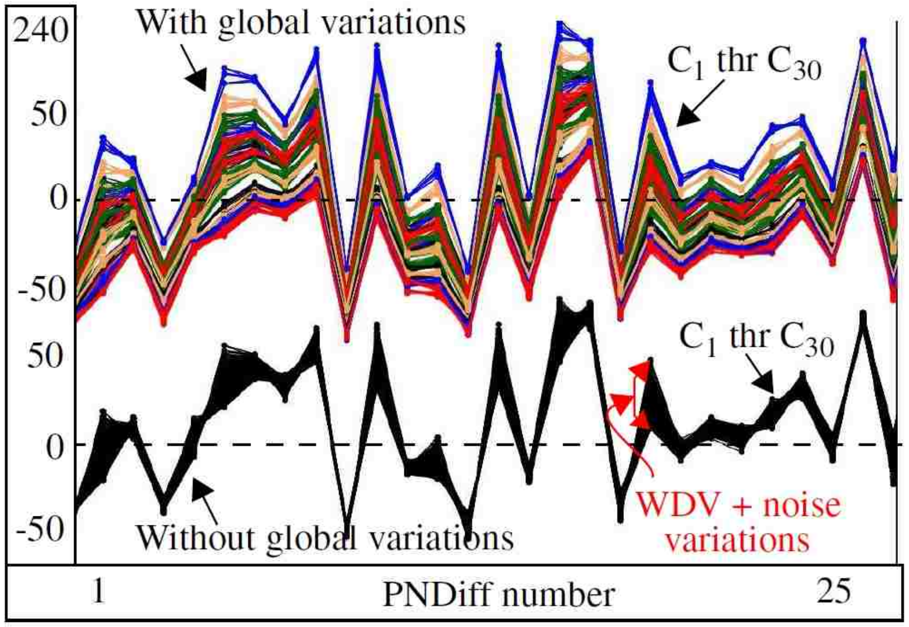


Fig. 3.5 TV compensated PNDiffs with (top) and without (bottom) global variations for 30 chips.

the top have been TV compensated as described in Section 4.4.2, i.e., using the enrollment values for each chip as the ‘reference’. The vertical offsets between the waveform groups are caused by global (chip-wide) variations, i.e., variations in the overall performance characteristics of the chips. Although global variations can be leveraged as a source of entropy, similar to within-die variations, there are drawbacks to depending on it.

To illustrate this problem, the black waveforms shown along the bottom of Fig. 3.5 are again from the 30 chips but are TV compensated using a special process in which the enrollment data from chip C_1 is used as the reference for all chips. This effectively eliminates the global variations and leaves only measurement noise, uncompensated TV noise and within-die variations (WDV) (see label in figure). In a large population of chips, it is highly likely that sets of chips will have the same level of global variations, so this graph illustrates this case, where only within-die variations can be leveraged as a source of entropy.

The magnitude of the noise sources is reflected in the width of the band of same colored waveforms shown along the top of Fig. 3.5 . Measurement noise (with 16 sample

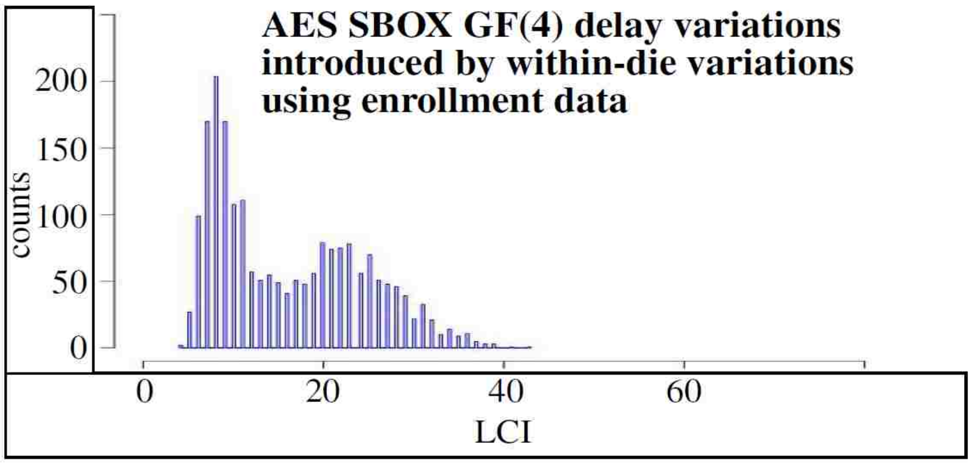


Fig. 3.6 Histogram of enrollment delay variations using TV compensation of PNDiffs with no global variations

averaging) is approx. 1 PN on average (approx. 18 ps), so the majority of the variation is introduced by uncompensated TV noise. The mean value of variation, computed as the mean of the 3σ values of the 10 TV compensated PNDiffs, that remains in the waveforms is on average approx. +/- 2.5 LCIs or 45 ps above or below the enrollment value, and the worst case value is less than +/- 8 LCIs or 145 ps. This number is important since it represents the amount of entropy that is lost, i.e., within-die variations less than this LCI value are more difficult to leverage. Within-die variations are reflected in the change in shape of the waveform groups for each chip. The magnitude of the variations introduced by within-die variations is, on average, approx. 4x larger (20 LCIs) than the average variation introduced by TV noise (5 LCIs), i.e., 360 ps vs 90 ps, resp.

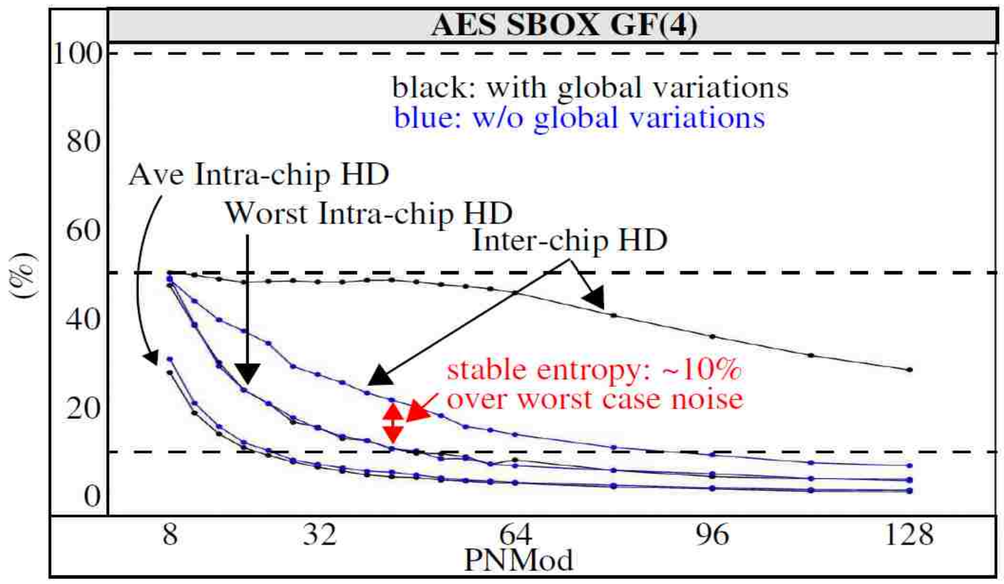


Fig. 3.7 Inter-chip HD and worst case and average case Intra-chip HD as a function of PN modulus.

A quantitative analysis of the entropy provided by within-die variations is shown in Fig. 3.6 using the 2048 PNDiffs from the 30 chips. The range across the 30 chips for each of the 2048 PNDiffs is computed using the TV compensated waveforms shown along the bottom of Fig. 3.5 , i.e., those without global variations. Only the enrollment PNDiffs are considered here, so the histogram plots the distribution of the 2048 ranges without TV noise. Given that measurement noise is very low, the shape of the histogram is predominated determined by within-die variations. As indicated above, the average value is close to 20 but the ranges vary from 10 to more than 40.

Fig. 3.7 provides a second quantitative analysis using the hamming distances (HD) of bitstrings computed using the proposed bitstring generation algorithm and the ModPNDiffs with and without global variations. The analysis is carried out over a set of *PN modulus* (**PNMod**) values plotted along the x-axis. Inter-chip HD is computed by counting the number of bits that are different in the 2048-bit bitstrings produced by two chips during enrollment and then dividing by the number of bits. The values plotted are the average Inter-chip HDs across all possible pairings of the bitstrings (30*29/2=435 pairings). Intra-chip HD is computed in a similar fashion except the pairings are defined using the bitstrings produced at the 10 TV corners for each chip (10*9/2=45 pairings). The value plotted is again the average computed across the 30 individual chip values. Worst-case Intra-chip HD is simply the maximum value produced by one of the individual chips.

The curves for worst case and average case Intra-chip HD in Fig. 3.7 reflect the noise levels, while the difference between the Inter-chip and Intra-chip HD curves reflect the range of usable entropy. The results *with* global variation included are shown in black while the results *without* global variations are shown in blue.

The bit flips created by uncompensated TV noise remains relatively constant independent of whether global variations are present or not, as shown by the superposition of the black and blue Intra-chip HD curves. The difference between the Inter-chip HD without global variations and the worst-case Intra-chip curves varies between 0% on the left to approx. 15% at the widest point around PNMod = 28. The worst-case Intra-chip HD at PNMod of 48 is approx. 10% while the Inter-chip HD is approx. 20%. This suggests that the average Inter-chip HD of a large chip population will be smaller than its ideal value of 50% without some type of entropy amplification process. The Inter-chip HD with global variations shows that the ideal value of 50% is nearly achieved for PNMods up to approx. 64. Unfortunately, as just mentioned, this is not likely to hold true as the number of chips used in the HD calculation increases well beyond the 30 available in our experiments. Therefore, in these experiments and on this 28 nm FPGA, either entropy amplification methods or other sources of entropy need to be leveraged to produce good quality bitstrings.

3.5.2 AES SBOX GF(26), Standard vs. WDDL

The test setup for the Standard GF(256) and WDDL versions of the AES SBOX is similar to that shown in Fig. 3.4. As indicated above, the structure of the Standard version is un-constrained and therefore, is subject to static and dynamic hazards occurring internally and on some outputs, which act to increase the occurrence of bit flips.

Fig. 3.8 (a) presents the statistical HD results in the same fashion as discussed in relation to Fig. 3.7 . The results are very similar to the GF(4) version except for the approx. doubling of the worst- and average-case Intra-chip HD over the GF(4) version. The increase in bit flips is directly attributable to presence of glitching. Note that glitching can increase both Intra-chip and Inter-chip HD. For paths whose delays are affected by glitches consistently across all TV corners, the effect is beneficial because the path delay typically changes by 10 to 100 LCIs, and therefore represents a significant source of within-die variations. For those paths where the glitch is present at some TV corners and disappears at others, the effect is detrimental, resulting in bit flips. The worst-case Intra-chip HD and Inter-chip HD curves illustrate that both types occur because the distance between the curves (and their shape) is similar to the corresponding curves shown in Fig. 3.7 . Although Inter-chip HD increases, this benefit is partially offset by the increase in worst-case bit-flips. Average-case Intra-chip HD, on the other hand, only increases slightly. Although we cannot present the results in detail here, it turns out that a small subset of our chips have many more occurrences of the detrimental form of glitching than the remaining chips. It was also possible to identify these glitchy chips by the difference in their rising and falling delays as shown in Fig. 3.2 (a), using data from the WDDL version of the AES SBOX. The falling PNs (blue waveforms) are offset downwards from the rising PNs (black waveforms) in the extra glitchy chips, i.e., the falling delays are noticeable smaller than the rising delays. The extra glitchy chip Intra-chip HDs are 3 times larger than the less glitchy chips.

3.5.3 Margin Technique

Fig. 3.8 (b) shows the results after applying a **Margin technique**. The method identifies PNDiffs during enrollment that have the highest probability of introducing bit flips. The PN modulus technique illustrated in Fig. 3.2 shows several examples of bit flips that occur at data points 4, 10 and 14. All of these data points are close to the lines that represent the boundaries between ‘0’ and ‘1’, i.e, 0, 31 and 63. The Margin technique classifies an enrollment PNDiff as ‘invalid’ if it falls within a small region (a margin) around these boundaries. The margin is set ideally to the worst case TV noise level for best results, but can be tuned according to the level of tolerance the server has to bit flips. A helper data bitstring is constructed during enrollment that records the valid status of each PNDiff data point. The helper data is stored on the server along with the margin, PNMod, challenge and response bitstrings. During regeneration, the server sends the margin, PNMod, challenge and helper data to the token, which uses the helper data to discard the ‘weak’ bits in the response.

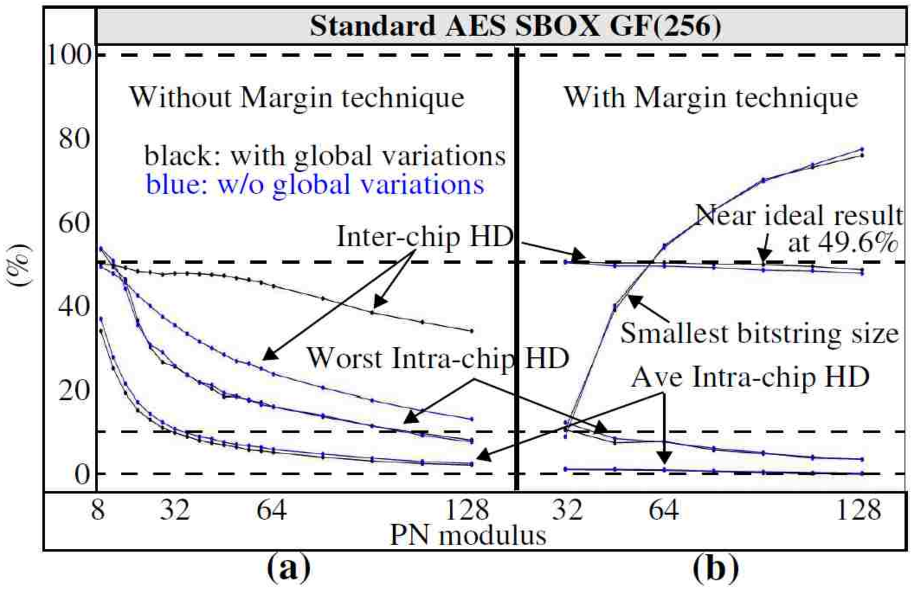


Fig. 3.8 Hamming distance (HD) results without (a) and with (b) the Margin technique for the Standard design.

The Margin technique significantly improves both the Intra-chip and Inter-chip HD results, as shown on the Fig. 3.8 (b). We used a Margin of 7 as the threshold to identify ‘weak’ bits in the response. Inter-chip HD improves because the PNDiffs corresponding to the generation of the ‘strong’ bits in different chips can now vary. This is true because within-die variations cause PNDiffs for some chips to fall within the margins, while on others, those same PNDiffs are outside the margins. Another important characteristic is the lower sensitivity of the results to whether global variations are present or not, which we indicated earlier is a highly desirable feature.

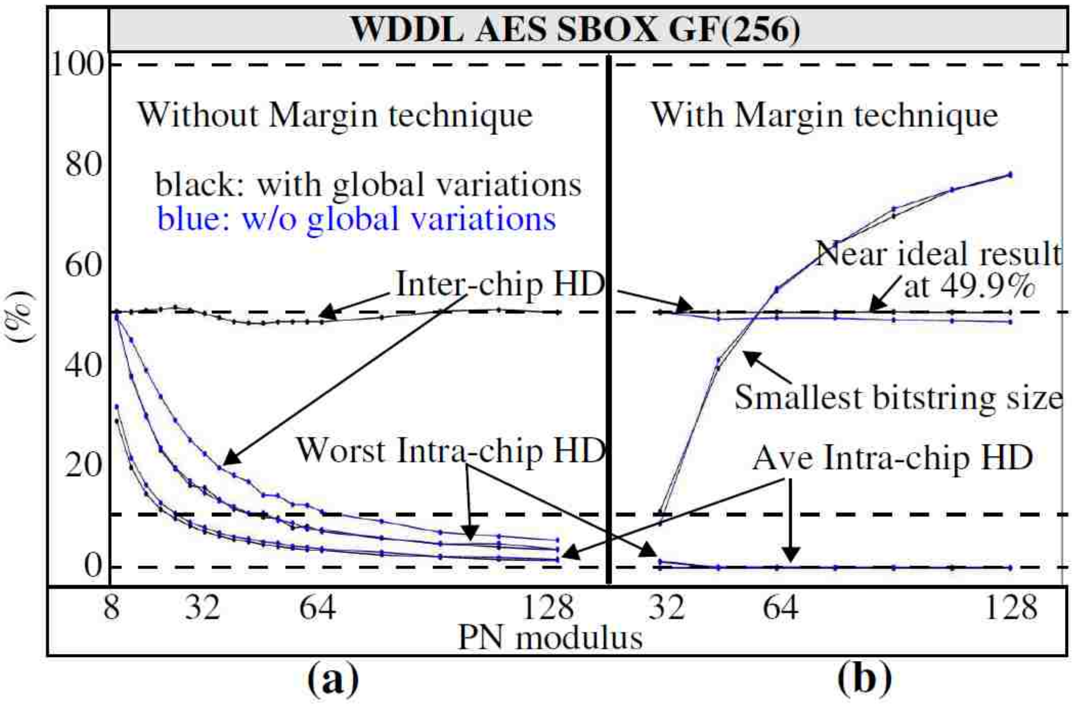


Fig. 3.9 Hamming distance (HD) results without (a) and with (b) the Margin technique for the WDDL design.

The size of the smallest bitstring generated by one of the 30 chips is also plotted in Fig. 3.8 (b) to illustrate the overhead associated with the helper data. By selecting a PNMod that is >= 64, the helper data bitstring is no larger than twice the size of the response bitstring in the worst case. It is also possible to use the complement of the helper data to generate a second response bitstring when the sum of the regions delineated by the margins is equal to the sum of the 'valid' regions defined for '0' and '1'. For example, a

PNMod of 64 as shown in Fig. 3.2 requires the margins to be set to 8, yielding valid regions of size 16. The second response bitstring uses the same set of PNDiffs but first adds an offset equal to 1/4 of the PNMod (16 in the example) before applying the modulus operation, which effectively shifts the distribution and converts all of the previous ‘weak’ bits into ‘strong’ bits (and vise versa), thereby making the helper data to response data ratio 1.

The results using the WDDL version are shown in Fig. 3.9 . The longer paths present in the WDDL version are responsible for the improvement in the Inter-chip HD to nearly ideal as shown on the left side in Fig. 3.9 (a). We confirmed this in a separate set of experiments (not shown) in which the path lengths in the Standard version are doubled. Therefore, longer paths improve Inter-chip HD but only in the case where global variations are preserved, i.e., the Inter-chip HD curve without global variations shows a very different result. The results using the Margin technique shown in Fig. 3.9 (b), on the other hand, are nearly ideal with or without global variations. The Intra-chip HD curves also illustrate that the majority of the bit flips that remain in the corresponding results from Fig. 3.8 (b) are attributable to the glitches produced in the Standard version, i.e., margining is not effective for glitches because the change in delay is larger than the worst case TV noise used as the margin. This is evident by the near 0 values for the worse case and average Intra-chip HD for the WDDL version.

3.5.4 NIST Statistical Test Results

The enrollment bitstrings generated in each of these 8 experiments were used as input to the NIST statistical test suite [60]. The small size of the bitstrings (largest is 2,048 bits), allowed up to 10 of the 15 NIST tests to be applied. The test is classified as passed if at least 28 of the 30 chip bitstrings pass the test. The NIST results are similar for the four sets of results in Fig. 3.8 (a), where all tests are passed for PNMod values less than 64. The PNMod of 64 represents a cut-off where some tests are failed but by only 2-3 chip in the worst case. The fail rates increase for PNMods larger than 64, with only a few passing some of the tests at the largest PNMod values. In contrast, the NIST results for the WDDL experiments shown in Fig. 3.9 are good throughout the entire PN modulus range, with only a few instances of fails, and by only 3 chips in the worst case. These results suggest that glitchy implementations of the FU produce bitstrings of good statistical quality but impose restrictions on the PNMod values, while glitch-free FUs are able to produce high quality bitstrings under a wider range of modulus values.

3.5.5 ATPG Analysis of Entropy

We used CADENCE Encounter Test (ET) to analyze the number of paths in the WDDL version of the AES SBOX. The underlying entropy source consists of both individual LUT gate delays and the interconnect routing delays, which are combined in unique ways and measured as path delays by HELP. Therefore, the number of paths reflects the amount of entropy present in the functional unit. This analysis will help support our claim that HELP is a strong PUF, with both a large input and output space, when used with functional units in which the number of paths is exponentially related to the number of its inputs.

A WDDL implementation contains two networks of interconnected logic gates (true and complemented) that ‘cross-over’ at points where inverters occur in the original network. The RC synthesized AND-OR-NOT version of the AES SBOX (see Fig. 3.3) produced 26 NOT gates in a network of 570 total gates, so the number of cross-overs is

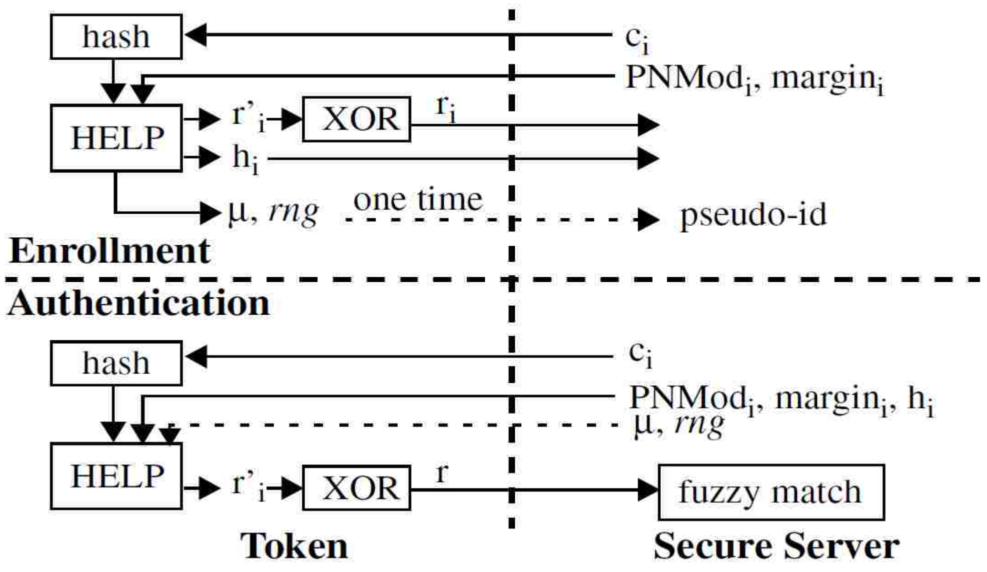


Fig. 3.10 Proposed authentication protocol.

fairly limited. With 16 inputs, the expected number of paths would be 2^16 or 65,536. ET reports 15,511 structural paths, which reflects the small interconnection structure between the two networks. As expected, automatic test pattern generation (ATPG) reports that 98.6% of all paths are hazard-free robust testable, which indicates that the paths are independent. Using the set of 512 WDDL vectors (Section 4.4.3), 37.8% of these paths are tested, which indicates that the remaining paths can only be tested by violating the complementary input patterns required with WDDL. However, testing the WDDL implementation using illegal patterns is possible and recommended when operating the functional unit in PUF mode.

3.6 Authentication Protocol

The proposed authentication protocol is shown in Fig. 3.10 . During enrollment, the server generates random challenges, c_i, $PNMod_i$ and $margin_i$ which are used by the token as a seed to an LFSR (or a pair of LFSRs to enable arbitrary two vector sequences to be applied). The PUF produces response r_i and helper data h_i, which are stored on the server with the challenge information. In cases where global variations are utilized, a μ and *rng* are also computed for the chip and stored on the server (note these values can also be used as a pseudo-id for the chip). The challenge is optionally passed through a cryptographic hash function to increase the difficulty of model building attacks which attempt to systematically apply a set of seeds designed to carry out path delay tests in a deterministic manner. The hash makes it difficult to determine how to choose c_i such that the output of the hash is controlled to specific seed values. The XOR obfuscation function of the response is optionally added for a similar purpose (note that only one of the input and output obfuscation methods is needed). As indicated in [39], XOR networks amplify bit flip behavior in *r* and therefore, are applicable only when Intra-chip HDs are very low. Authentication is carried out in a similar fashion except for the direction of transmission of the helper data, h_i, μ and *rng*. Note that μ and *rng* are not needed if the PNDiffs are TV compensated to a universal standard (which also eliminates entropy from global variations).

As indicated, the margin and PNMod parameters are also beneficial because they expand the CRP space. However, allowing these parameter to be set without constraints can be used by an adversary to assist with model building. Our experiments suggest that a hard coded margin or allowing only a small range of values, e.g., between 5 and 8, accomplishes the goal of improving the statistics while maintaining a limited information leakage channel. The same is true of the PNMod parameter, where only a limited set of values should be allowed, e.g., restricting to powers of 2 also significantly simplifies the implementation of the modulus operation while providing a 'limited' expansion of the CRP space.

3.7 Summary and Conclusion

In this chapter, we investigated the strengths and weaknesses of using a delay-based strong PUF for authentication. Glitch-free functional units were used as the entropy source and shown to enhance the quality of the generated bitstrings. Within-die variations by itself is not large enough to produce unique bitstrings across a large population of chips. A margining technique is shown to significantly improve the statistical quality of the bitstrings while adding moderately to the storage overhead in the secure database.

The following areas will be investigated in future work. We will investigate the use of ATPG generated input vectors as challenges, which can target additional sources of entropy represented by 'random pattern resistant' paths, that are not likely tested using an LFSR scheme. We will also investigate enrollment schemes which store PNDiffs directly through a one-time interface. These 8-bit values can then be used to generate a set (>> 8) of bitstrings by changing the modulus and margin parameters, thereby improving the storage efficiency on the server. Alternative, lower overhead, glitch-free logic implementation styles will be investigated as an alternative to WDDL. Low power techniques that only reduce the occurrence of glitches will also be investigated.

Although not reported on in this paper, we have also evaluated a voltage-based enrollment (VBE) scheme, which uses the bitstrings generated at a fixed set of supply voltages, in particular, those at the extremes of the specification range, and then records, as weak bits in the helper data, those bits that flip in the regenerated bitstrings. VBE works well to reduce the Intra-chip HD for normally synthesized functional units, i.e., those with glitches. We also found significant diversity is created by the synthesis tool in path delays and the corresponding bitstrings when inconsequential changes are made to the HDL, which again can be used to expand the input/output space of HELP. Last, we are investigating the applicability of techniques described here to board-level authentication as described in [35].

CHAPTER 4
Leveraging Distributions in Physical Unclonable Functions

A special class of Physical Unclonable Functions (PUFs) referred to as strong PUFs can be used in novel hardware-based authentication protocols. Strong PUFs are required for authentication because the bitstrings and helper data are transmitted openly by the token to the verifier and therefore, are revealed to the adversary. This enables the adversary to carry out attacks against the token by systematically applying challenges and obtaining responses in an attempt to machine-learn and later predict the token's response to an arbitrary challenge. Therefore, strong PUFs must both provide an exponentially large challenge space and be resistant to machine-learning attacks in order to considered secure. We investigate the security properties of a Hardware-embedded Delay PUF called HELP in this paper. HELP leverages within-die variations in path delays within a hardware-implemented macro (functional unit) as a random source of information for bitstring generation. Several features of the HELP processing engine significantly improve its resistance to model-building attacks. Most important is a novel linear transformation proposed within the HELP processing engine for dealing with changes in delay introduced by adverse temperature-voltage (environmental) variations. The technique also increases entropy by making the measured path delay values dependent on the other values included in the distribution used to generate the entire bitstring.

4.1 Introduction

A physical unclonable function (PUF) is a next-generation hardware security primitive. Security protocols such as authentication and encryption can leverage the random bitstring and key generation capabilities of PUFs as a means of hardening vulnerable mobile and embedded devices against adversarial attacks. Authentication is a process that is carried out between a hardware token (smart card) and a verifier (a secure server at a bank) that is designed to confirm the identities of one or both parties [62]. With IoT, there are a growing number of authentication applications in which the hardware token is resource-constrained. Conventional methods of authentication which use area-heavy cryptographic primitives and non-volatile memory (NVM) are less attractive for these types of evolving embedded applications [63]. PUFs, on the other hand, can address issues related to low cost because they can potentially eliminate the need for NVM. Moreover, the special class of *strong PUFs* can further reduce area and energy overheads by reducing the number and type of cryptographic primitives and operations.

A PUF extracts random information (entropy) from variations in the physical and electrical properties of ICs, that are unique to each IC, as a means of generating digital secrets (bitstrings). The bitstrings are generated on-the-fly when needed, and are reproducible under a range of environmental variations. The ability to control the precise generation time of the secret bitstring and the sensitivity of the PUF entropy source to invasive probing attacks (which act to invalidate it) are additional attributes that make them attractive for authentication in hardware tokens.

Most proposed PUF architectures require the insertion of a dedicated array of identically-designed test structures and are classified as *weak PUFs* because of their limited number of challenge-response-pairs (CRPs). Authentication applications reveal both the challenges and responses during authentication operations and therefore weak PUFs are not suitable. *Strong PUFs*, on the other hand, are able to produce an exponential number of challenge-response-pairs (CRPs) but in order to be considered secure, they must be resistant to model-building attacks.

The hardware-embedded Delay PUF (HELP) analyzed in this paper generates bitstrings from delay variations that occur along paths in an on-chip macro, such as a cryptographic primitive. Although it is possible to construct a hardware instantiation of the functional unit which possesses an exponential number of paths, in an attempt to meet strong PUF requirements, this is not necessary for HELP for several reasons. First, unlike other strong PUFs, the task of generating challenges, i.e., test vectors that test all the paths in a moderately complex functional unit, is non-trivial. Although this is a one-time cost for a specific implementation, it still represents a significant additional burden for the adversary.

Second, the HELP processing engine defines a set of *user-defined parameters* which are used to transform the measured path delays into bitstring responses. One of these parameters, called the *Path-Select-Mask* provides a mechanism to choose *k* paths from *n* that are produced, which enables an exponential number of possibilities. Therefore, the *Path-Select-Mask* allows the set of path delays used in the bitstring generation process (the distribution) to vary from one authenticaton request to the next. This feature when combined with a second processing step within the HELP engine called *Temperature-*

Voltage-Compensation (**TVCOMP**), can introduce changes in the bit value produced by a specific path delay. **In other words, the mapping between path delays and bits in the response is a function related to the values of other components of the delay distribution**.

This is a novel and important source of entropy that is only possible by introducing a transformation that factors in the behavior of the entire distribution used for bitstring generation. It should be noted that this type of distribution-based entropy-enhancing method is not applicable to PUFs which generate 'soft data' that is designed to be identical, e.g., RO PUFs [64], because the characteristics of the distribution are invariant. The path delays used by HELP vary widely in length, and therefore constructing distributions with different *means* and *standard deviations* is easy to do.

This paper is dedicated to showing this 'distribution-effect' on bitstring diversity. The implications are two fold. First, HELP can make use of smaller functional units, i.e., those without an exponential number of paths, and still achieve an exponential number of challenge-response-pairs (CRPs) as required of a strong PUF. Second, the difficulty of model-building HELP using machine learning algorithms will be more difficult because the path delays of the physical model are no longer constant.

4.2 Related Work

Although references [65] and Error: Reference source not found describe previous research on HELP, no prior work exists that describes the security properties of HELP and the analysis presented in this paper. A method to estimate the "extractable" entropy in

PUF-generated bitstrings is proposed in [67] by calculating the mutual information between the bias measurements done at enrollment and regeneration. The authors in [68] evaluate the robustness and unpredictability of five different PUFs (including arbiter, RO, SRAM, flip-flop and latch PUFs) by estimating the entropy from the available responses.

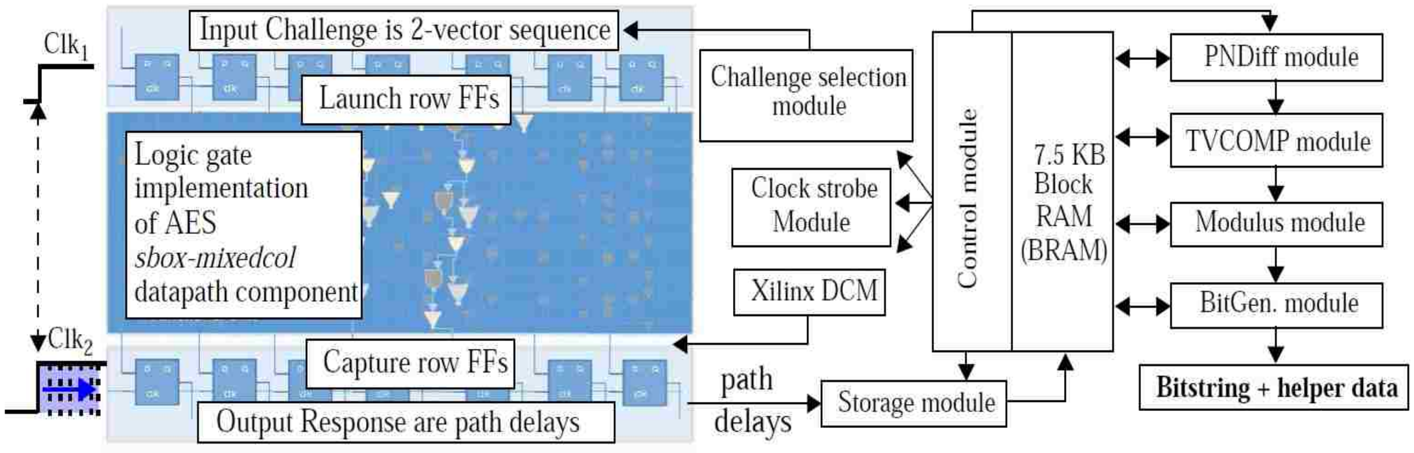


Fig.4.1 Instantiation of the HELP entropy source (left) and HELP processing engine (right).

4.3 HELP Overview

HELP attaches to an on-chip module, such as a hardware implementation of the cryptographic primitive, as shown in Fig.4.1. The logic gate structure of the functional unit defines a complex interconnection network of wires and transistors. The functional unit shown in Fig.4.1 is a 32-bit column from Advanced Encryption Standard (AES) which includes 4 copies of the SBOX and 1 copy of the MIXEDCOL (called **sbox-mixedcol**) [69]. This combinational data path component is implemented in a WDDL logic style [70], which doubles the number of primary inputs (PIs) and primary outputs

(POs) to 64. The implementation of sbox-mixedcol requires approx. 3000 LUTs on a Xilinx Zynq FPGA and provides approx. 8 million paths. Although the analysis carried out in this paper uses sbox-mixedcol, the results suggest that smaller functional units, i.e., less than 1000 LUTs, can be used as an alternative.

HELP accepts challenges in the form of 2-vector binary sequences. The vector sequences are applied to the PIs of the functional unit and the delays of the sensitized paths are measured at the POs. Path delay is defined as the amount of time (Δt) it takes for a set of 0-to-1 and 1-to-0 transitions introduced on the PIs to propagate through the logic gate network and emerge on a PO. HELP uses a clock-strobing technique to obtain high resolution measurements of path delays as shown on the left side of Fig.4.1. A series of launch-capture operations are applied in which the vector sequence that defines the input challenge is applied repeatedly using the Launch row flip-flops (FFs) and the output responses are measured using the Capture row FFs. On each application, the phase of the capture clock, Clk_2, is incremented forward with respect to Clk_1, by small Δts (approx. 18 ps), until the emerging signal transition is successfully captured in the Capture row FFs. A set of XOR gates connected between the inputs and outputs of the Capture row FF inputs (not shown) provide a simple means of determining when this occurs. When an XOR gate value becomes 0, then the input and output of the FF are the same (indicating a successful capture). The first occurrence in which this occurs during the clock strobing operation causes the current phase shift value to be recorded as the digitized delay value for this path. This operation is applied to all POs simultaneously.

The phase shifting module for Clk_2 is shown on the right side of the functional unit in

Fig.4.1. On-chip digital clock managers (DCMs) are commonly included in FPGA architectures. For example, Xilinx FPGAs typically incorporate at least one DCM with a digitally controlled *fine phase shift* control mechanism even on their lowest cost FPGAs.

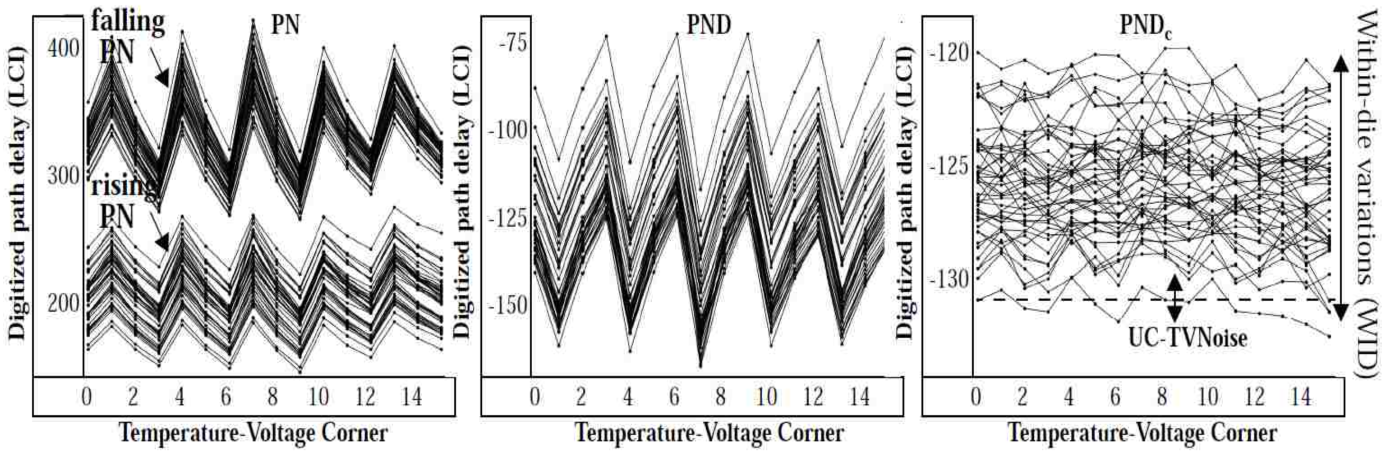


Fig.4.2(a) Example rising and falling path delays (PN), (b) Rise-fall path delays (PND)and (c) TV Compensated PND$_c$ for 45 chips (individual curves) and 16 TV corners (points in curves).

The digitized path delays are collected by a *storage* module and stored in an on-chip block RAM (BRAM) as shown in the center of Fig.4.1. A *Path-Select-Mask* is also sent by the verifier, along with the challenges, to allow path delays to be selected from those that are produced. Each digitized timing value is stored as a 15-bit value, with 11 binary digits serving to cover a signed range between +/- 1024 and 4 binary digits of fixed point precision to enable up to 16 samples of each path delay to be averaged. The 7.5 KByte BRAM allows 4096 path delays to be stored. We configure the applied challenges and masks to test 2048 paths with rising transitions and 2048 paths with falling transitions. The 15-bit digitized path delays are referred to as **PN**.

4.3.1 PN Processing

Once the PNs are collected, a sequence of mathematical operations are applied as shown on the right side of the Fig.4.1 to produce the bitstring and helper data. The *PNDiff* module creates unique, pseudo-random pairings between the rising and falling PN using two seeded linear feedback shift registers (LFSR). The two 11-bit *LFSR seeds* are user-defined parameters. PN differences, referred to as **PND**, are defined as (rising PN - falling PN), and are stored in the lower 2048 memory locations of the BRAM, overwriting the original set of rising PN.

Fig.4.2(a) shows an example of this process using two groups of 45 curves, one curve for each Xilinx Zynq 7020 chip that was tested. The curves shown along the bottom depict the PN from rising transition tests and those along the top from falling transition tests. The 16 line-connected points associated with each curve represent the chip's PN measured under a range of environmental conditions, called temperature-voltage (TV) corners. The PN at the x-axis position given by 0 are those measured under nominal (**enrollment**) conditions, i.e., at 25^{o}C, 1.00V. The PN at positions 1, 2 and 3 are also measured at 25^{o}C but at supply voltages of 0.95, 1.00 and 1.05 V. Similarly, the other groups of 3 consecutive points along the x-axis are measured at these supply voltages but at temperatures -40^{o}C, 0^{o}C, 85^{o}C and 100^{o}C. TV corners 1 to 15 are referred to as **regeneration** corners. Fig.4.2(b) plots the corresponding PND

As indicated earlier, TV-related effects on delay negatively impacts bitstring reproducibility. We propose a *TV compensation* (TVCOMP) process to reduce variations in the PND introduced by changes in TV conditions (called TV noise). The goal is to define a transformation that eliminates the saw-tooth behavior in the curves shown in Fig.4.2(b), making them as flat and straight as possible.

TVCOMP is applied to the entire set of 2048 PND measured for each chip during regeneration (note, Fig.4.2(b) shows only one of the PND from the larger set of 2048 that exist for each chip and TV corner). The TVCOMP procedure first converts the PND to ‘standardized’ values. Equation (4.1) represents the first transformation which makes use of two constants, μ_{test} and Rng_{test}, obtained from a histogram distribution of the measured PND.

$$zval_i = \frac{(PNDiff_{test} - \mu_{test})}{Rng_{test}} \tag{4.1}$$

$$PNDc_i = zval_i \cdot Rng_{ref} + \mu_{ref} \tag{4.2}$$

The second transformation is represented by Equation(4.2), which translates the standardized *zvals* to a new distribution with mean μ_{ref} and range Rng_{ref}. The μ_{ref} and Rng_{ref} constants are also user-defined parameters of the HELP algorithm.

The transformation carried out by TVCOMP is depicted in Fig.4.2(b) and Fig.4.2(c). The data in Fig.4.2(c) is obtained by applying TVCOMP procedure to the 2048 PND measured under each of the 16 TV corners for each chip, i.e., 16 TV corners * 45 chips = 720 separate applications. Since the same μ_{ref} and Rng_{ref} are used for all transformations, TVCOMP eliminates both TV noise and chip-wide performance differences between the chips.

The variations that remain in the PND_c shown in Fig. Fig.4.2(c) are those introduced by within-die variations (**WDV**) and *uncompensated* TV noise (**UC-TVNoise**). The UC-

TVNoise component of the data shown in Fig.4.2(c) is represented by the *y*-dimensional variations that occur **in each curve** (the worst case is approx. 3 which translates to approx. 50 ps). In general, bit-flip errors become more likely for PND_c with larger UC-TVNoise components. WDV also manifests as *y*-dimensional variations, but in this case, is represented by the **spread of the curves**. The spread is approx. 12 for the curves shown in Fig.4.2(c). Larger values of WID improve the statistical properties of the generated bitstrings, including randomness and uniqueness, and is therefore a desirable characteristic.

The HELP algorithm shown in Fig.4.1 implements TVCOMP by constructing a histogram distribution in the upper 2048 memory locations of the BRAM using the 2048 PND stored in the lower portion and then parses the distribution to obtain μ_{TVx} and Rng_{TVx}. Once the distribution constants are available, the PND in the low portion of the BRAM are converted to PND_c.

The last operation applied to the PN is represented by the *Modulus* operation shown on the right side of Fig.4.1. Modulus is a standard mathematical operation that computes the positive remainder after dividing by the modulus. The Modulus operation is required by HELP to eliminate the path length bias that exists in the PND_c. The value of the Modulus is also a user-selectable parameter, similar to the *LFSR seeds*, μ and *Rng* parameters. The HELP engine shown in Fig.4.1 overwrites the PND_c after applying the Modulus. The final values, called **modPND$_c$**, are used in the bitstring generation process.

4.3.2 Bitstring Generation

The bitstring generation process uses a fifth user-specifiied parameter, called the *Margin*, as a means of further improving the reliability of the bitstring regeneration process (beyond that provided by the TVCOMP process). Fig.4.3 illustrates the bitstring generation process using two sets of 18 $modPND_c$ from $Chip_1$ labeled $MaskSet_A$ and $MaskSet_B$[1] . A modulus of 20 is used in combination with a set of margins of size 3 surrounding two strong bit regions of size 6. Designators along the top given as ‘s’, ‘w’ classify each of the enrollment data points as either a strong or weak bit. Data points that fall on or within the hatched areas are classified as weak as a mechanism to avoid bit flip errors introduced by UC-TVNoise that occurs during regeneration.

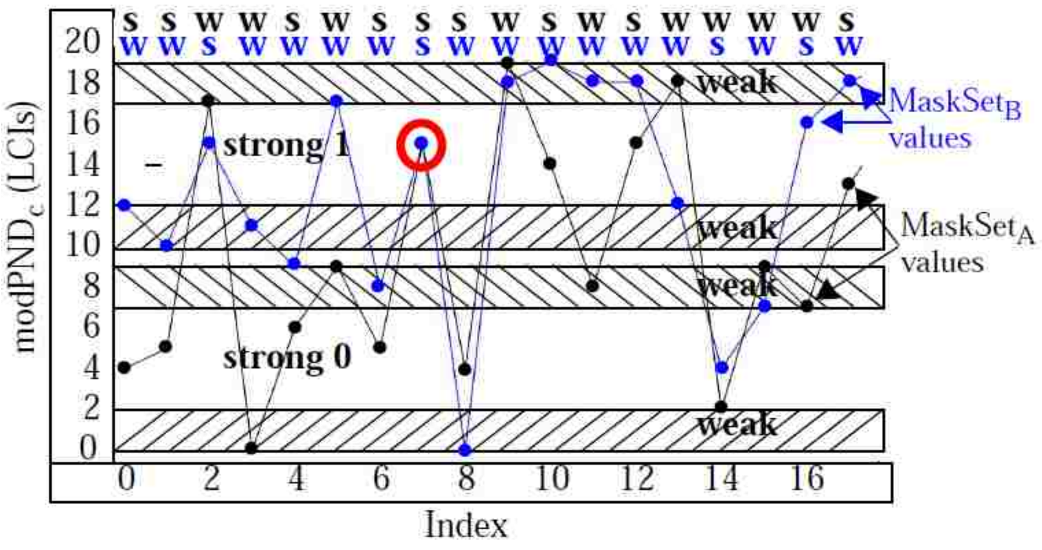


Fig.4.3Illustration of the Modulus-Margin process carried out by HELP for bitstring generation.

1 The reason we include two sets will be explained later.

The Margin method improves bitstring reproducibility by eliminating data points classified as ‘weak’ in the bitstring generation process. For example, 9 of the data points from MaskSet_A are strong and 4 are strong from MaskSet_B. The remaining data points are classifed as weak because they are too close to the bit-flip lines of 0, 9, 10 and 19. A helper data bitstring is generated to record the status of the bits. When HELP is used in authentication protocols, the helper data bitstring and strong bitstrings are sent to the verifier as proof-of-identity.

4.4 Security Analysis

In this section, we investigate several important security properties of HELP that relate to its resistance to model building and to the number of bitstrings that each token can generate using the six user-defined parameters described earlier, i.e., *Modulus*, *Margin*, μ_{ref} *and* Rng_{ref}, two 11-bit *LFSR seeds* and the *Path-Select-Mask*. We make the following assumptions in our security analysis:

- The adversary is not able to configure the token to run in enrollment mode. Enrollment mode is only possible using a special FPGA programming bitstream, which is generated and securely stored by a trusted authority.
- The system containing the PUF provides countermeasures against physical attacks such as differential power analysis and fault injection.
- The adversary has physical access to the token and can activate the PUF an unlimited number of times to produce both a strong bitstring and helper data bitstring using arbitrary challenges and masks of his choice.

- The adversary can control the six user-defined parameters within the limits imposed by the PUF implementation. For example, although any arbitrary set of 11-bit *LFSR Seeds* and *Path-Select-Mask* can be chosen and applied, the *Margin* and *Modulus* as well as the μ_{ref} and Rng_{ref} are constrained to discrete ranges of values (discussed below).
- The adversary has full knowledge of the functional unit and implementation details, and can run timing simulations, typically using the worst case process-voltage-temperature model provided by the FPGA vendor.

4.4.1 Parameter-Based Bitstring Diversity

Due to the interaction of the user-defined parameters, we present a conservative lower-bound estimate on the number of possible parameter combinations, i.e., those that ensure the generated bitstrings are random, reliable and unique for each token. Note that the source of entropy is fixed in this sub-section to a set of 4096 PN (in contrast to the analysis presented in the next sub-section).

As discussed above, five of the user-defined parameters, namely, μ_{ref}, Rng_{ref} , *Modulus*, *Margin* and the two *LFSR seeds*, can be used to apply different transformations to the same set of PN as a means of achieving bitstring diversity. The two 11-bit *LFSR seed* parameters allow any of the 2048 rising edge PN to be paired with any of the 2048 falling edge PN, yielding 4,194,304 possible combinations. Analysis of the data collected in our Zynq FPGA experiments indicates that the number of combinations of *Margins* and *Moduli* that yield high reliability (bit flip probability $< e$) is 20 (using Moduli 16 to

30 and Margins 3 to 5). The number of different μ_{ref} and Rng_{ref} parameters is conservatively estimated to be 10 each. Therefore, a total of 4,194,304 * 20 * 10 * 10 ~= 8.3 billion combinations of these five user-defined parameters are possible. This lower bounds the amount of effort required by an adversary in possession of the token to read out all the possible response bitstrings for a fixed set of 4096 PN, i.e., with the *Path-Select-Mask* set to select the same set of PN.

4.4.2 *Path-Select-Mask*-Based Bitstring Diversity

Unlike the parameter-based scheme, bitstring diversity introduced by the *Path-Select-Mask* is based on changing the underlying source of entropy. In other words, the 4096 PN are not fixed, but vary from one authentication to the next. The *Path-Select-Mask* is used by the server to select a subset *k* of *n* path delays produced by the set of applied challenges.

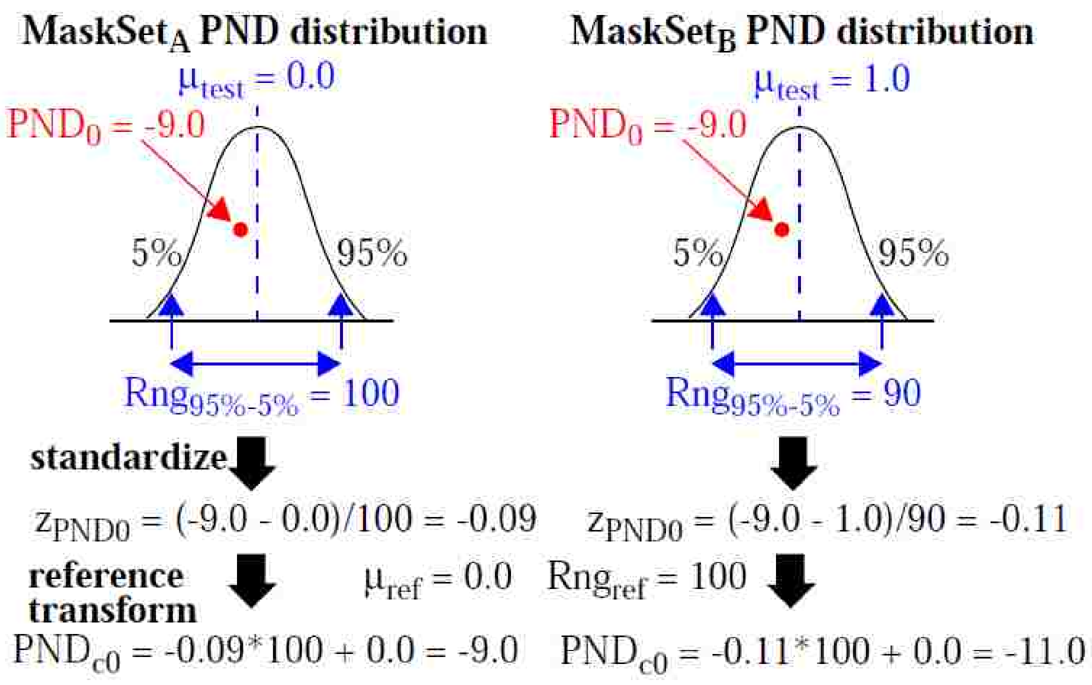
For example, assume that a sequence of challenges produces a set of 5,000 rising PN and a set of 5,000 falling PN, from which the server selects a subset of 2048 from each set. The number of ways of choosing 2048 from 5000 is given by Eq. (4.3). Therefore, the *Path-Select-Mask* enables an exponential *n*-choose-*k* PN selection process.

$$Path-select-combs=C_{2048}^{5000}=3.3\,e^{1467} \qquad (4.3)$$

Previous work has shown that an exponential number of response bits is necessary condition for a truly strong PUF but not a sufficient condition. The responses must also be largely *uncorrelated* as a means making it difficult or impossible to apply machine learning algorithms to model-build the PUF. The analysis provided in this section shows that the *Path-Select-Mask* in combination with the TVCOMP process provides additional entropy beyond that available in a fixed number of path delays. This is key to enabling HELP's entropy source to be relatively small while providing truly strong PUF characteristics in its output responses.

4.4.3 "Distribution-Effect" Bitstring Diversity

The set of PN selected by the *Path-Select-Mask*s changes the characteristics of the PND distribution, which in turn impacts how each PND is transformed through the TVCOMP process. The TVCOMP process was described earlier in reference to Eqs. (3.1) and (3.2). In particular, Eq. (3.1) uses the μ_{test} and Rng_{test} of the measured PND distribution to standardize the set of PND before applying the reverse transformation given by Eq. (3.2).



MaskSet$_A$ PND distribution
μ_{test} = 0.0
PND$_0$ = -9.0
5%
95%
Rng$_{95\%-5\%}$ = 100
standardize
z_{PND0} = (-9.0 - 0.0)/100 = -0.09
reference transform
μ_{ref} = 0.0
PND$_{c0}$ = -0.09*100 + 0.0 = -9.0
MaskSet$_B$ PND distribution
μ_{test} = 1.0
PND$_0$ = -9.0
5%
95%
Rng$_{95\%-5\%}$ = 90
z_{PND0} = (-9.0 - 1.0)/90 = -0.11
Rng$_{ref}$ = 100
PND$_{c0}$ = -0.11*100 + 0.0 = -11.0

Fig.4.4Impact of the TVCOMP process on PND0 when members of the PND distribution *change for different mask sets A and B.*

Fig.4.4 provides an illustration of the TVCOMP process. The two distributions are constructed using data from the same chip using two different sets of *Path-Select-Masks*, MaskSet_A and MaskSet_B. The μ_{test} and Rng_{test} of the MaskSet_A distribution are given as 0.0 and 100 while the values for the MaskSet_B distribution are given as 1.0 and 90. The point labeled PND_0 is present in both distributions but the remaining components (or a portion as discussed below) are different, which introduces differences in μ_{test} and Rng_{test}. Eq. (3.1) is used to *standardize* the PND's of both distributions, which translates PND_0 differently in each distribution, to values given by -0.09 and -0.11, resp. Eq. (3.2) translates the standardized values back into an integer range using a common μ_{ref} and Rng_{ref} given by 0.0 and 100, resp. The TVCOMP'ed PNDs (PND_{c0}) are given as -9.0 and -11.0. **This shows that the TVCOMP process introduces variations in PND_c even when they are generated from the same pairings of rising and falling path delays**. The change in a PND_c occurs because the relative position of its corresponding PND in each of the multiple distributions is dependent on the other members of the distribution.

The *Modulus-Margin* graph of Fig.4.3 described earlier provides an example using actual FPGA data from Chip C_1. The subset of modPND_c values shown are in fact computed from the same set of PN, but are included in PN distributions that are derived using different mask sets, MaskSet_A and MaskSet_B. Note that the changes introduced by the TVCOMP transformation are further obscured by the Modulus, making some points change by a little and others by a lot (and some not at all).

This 'distribution-effect' can be used by the verifier as a means of increasing the unpredictability in the generated response bitstrings. One strategy would be to purposely skew the μ_{test} and Rng_{test} through clever selection of PN to change the bit values generated by a set of PN that have been used in previous authentications.

4.5 Experimental Results

In order to determine the effectiveness of this type of approach, we construct a set of PN distributions for evaluations. In all distributions, we include a fixed set of 300 rising and 300 falling PN and draw the remaining 2048 - 300 = 1748 PN from a rising and falling 'Master' distribution. The remaining 1748 PN are confined to specific regions of the Master distribution as a means of systematically forcing changes to μ_{test} and Rng_{test}. The regions are called ***windows*** in the Master distribution.

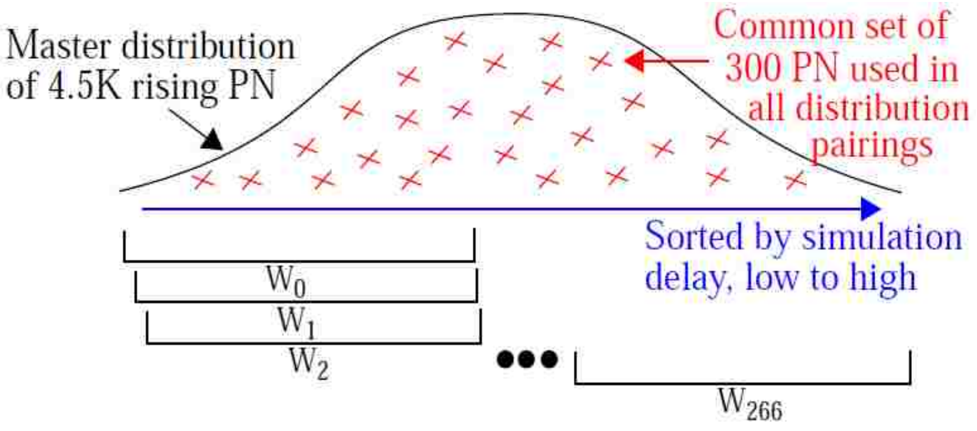

Master distribution of 4.5K rising PN
Common set of 300 PN used in all distribution pairings
Sorted by simulation delay, low to high
W_0
W_1
W_2
●●●
W_{266}

Fig.4.5 Master distribution with 4500 rising PN with sequence of 'windows' used to draw remaining components of rising PND components for a total of 2048.

Fig.4.5 illustrates the process used to create the distributions for the rising PN (the same process is used for the falling PN). The PN in the Master distributions are first sorted according to their worst-case simulation delays. The 300 common PN are randomly chosen across the entire Master distribution and are represented as ‘x’ in the

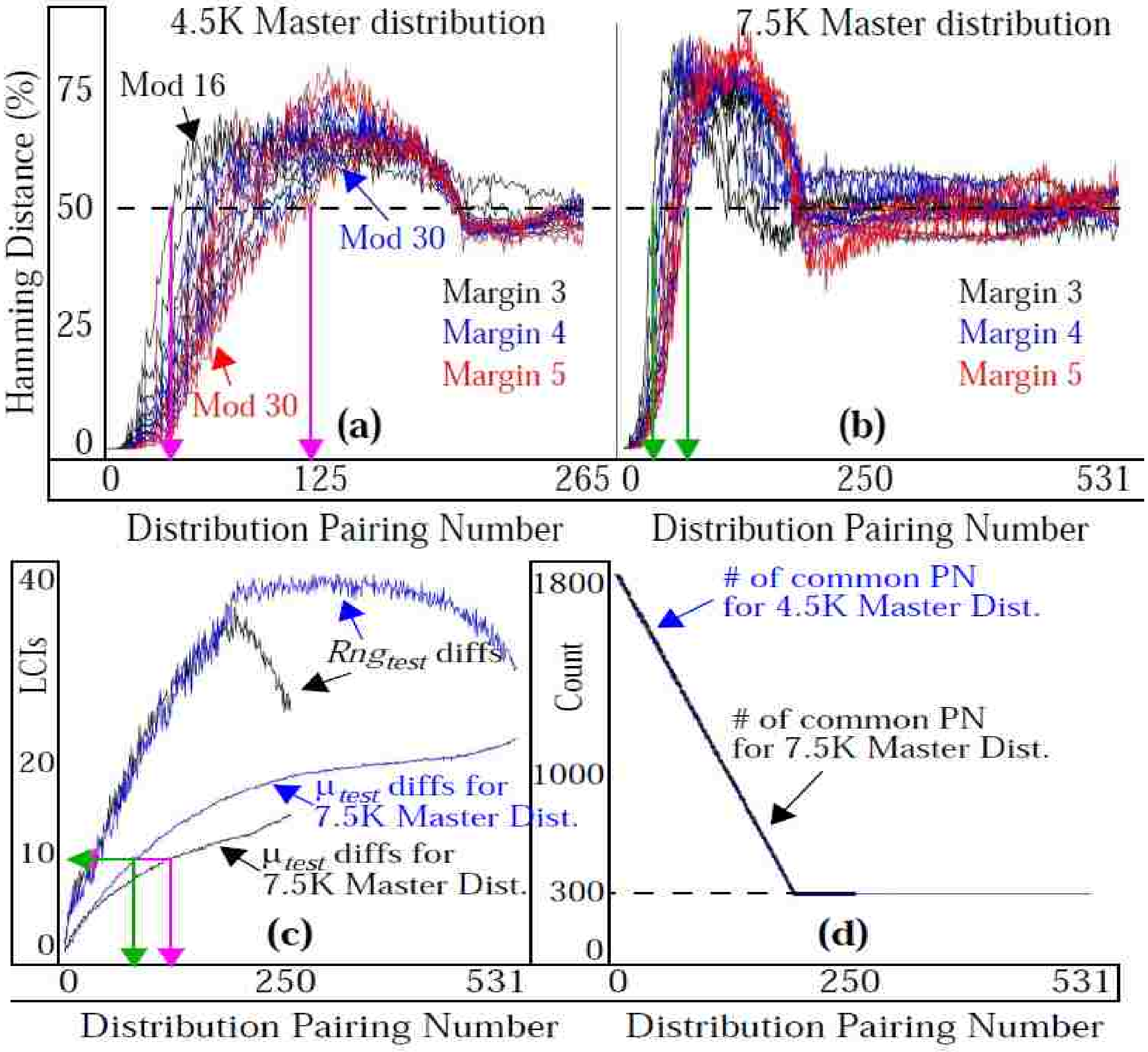


Fig.4.6 Hamming distance of strong bitstrings derived from distributions in which at least 300 of the modPND$_c$ values are common in each pair of distributions of size 2,048. The HD is computed using portions of the bitstrings derived from various pairings of

distributions, but using only the set of bits corresponding to modPND$_c$'s that are common in both distributions and are identified as strong in both helper data bitstrings of the pair. (a) Results using first Master distribution over 20 combinations of Margins and Moduli, (b) same using second Master distribution, (c) change in μ_{test} Rng_{test} between the PND distribution pairs (d) the number of common PND in each PND distribution pair.

figure. A series of windows, labeled W_i, are then created that contain 2000 consecutive elements from the 4500 elements in the Master distribution. The remaining 1748 components are drawn from within each of these windows of 2000, excluding PN that are members of the original 300. Each consecutive window is skewed to the right by 10 elements. A total of 267 distributions of 2048 PN are constructed in this fashion. A PND distribution for W_i is constructed from the corresponding rising and falling W_i PN distributions. The common set of 300 rising and falling PN are matched in the same fashion in all PND distributions to create identical PND values. The analysis presented below creates a set of 266 pairing of PND distributions where each of the W_1 through W_{266} PND distributions are paired with the W_0 distribution. The different between the μ_{test} and Rng_{test} values of the two PND distributions increases for successive pairings, allowing the distribution-effect to be evaluated systematically. The TVCOMP, Modulus and bitstring generation processes are applied to the PND distributions. The differences in the bitstrings from each PND pairing is computed using *Hamming distance* (HD). **We use only those bits corresponding to the common set of 300 in the HD calculation**. draw remaining components of rising PND components for a total of 2048.

Note that the number of PND that are common in each pairing of distributions is larger than the forced 300 in cases where W_i overlaps with W_0. This happens because random sampling within the overlapping windows chooses additional PN that are common to both distributions. Additional common rising PN are deliberately paired with additional common falling PN as a means of creating worst case conditions (the more commonality that exists, the smaller the change in the μ_{test} and Rng_{test} of the two distributions, which in turn reduces the bitstring HD).

The results shown in Fig.4.6 are computed using data collected from 45 copies of a Xilinx Zynq 7020 chip. Fig.4.6(a) and (b) show the HD results for two different Master distributions, one that contains 4.5K rising and falling PN and one that contains 7.5K rising and falling PN. The larger Master increases the number of window pairs depicted in Fig.4.5 to 531.

The HDs are computed using bits under the condition that both bits of the pair are strong. This condition is illustrated in Fig.4.3 for the modPND$_c$ corresponding to bit 7 (which is circled). Eq. (4.4) gives the expression for HD.

$$HD_{W_i} = \frac{\sum_{j=0}^{NumChips} \sum_{k=0}^{NumStrongPairs} b_{W_{0,} j,k} \, xor \, b_{W_{0,} j,k}}{NumChips \times NumStrongPairs} \tag{4.4}$$

The HDs are reported as percentages by dividing the number of strong bit pairing differences by the total number of strong bit pairings. The W_0-W_i pairings are plotted along the x-axis in Fig.4.6(a) and (b). The HD curves for 20 different combinations of the Margins and Moduli are superimposed to illustrate that the trend is similar.

The HDs are zero for cases in which W_0-W_i have significant overlap (left-most points) because the μ_{test} and Rng_{test} of the two distributions are nearly identical under these conditions. As the windows separate, the HDs rise quickly to 50%, especially for smaller Moduli. The smallest and largest shift required to reach 50% is highlighted with vertical lines for the 4.5K and 7.5K Master distributions, resp. Fig.4.6(c) plots the average difference in the μ_{test} and Rng_{test} of the distribution pairs. Note that the 7.5K Master distribution achieves the ideal result of 50% for smaller shifts in μ_{test} and Rng_{test}, which in turn allows more flexibility in choosing the PN to achieve the 'distribution-effect'. Fig.4.6(d) plots the average number of common PN used in the HD calculation. As indicated earlier, the number is larger than the 'forced' 300 for overlapping windows. The 'overshoot' of the HDs above the ideal of 50% is difficult to pin down. Interestingly, this correlation is less significant beyond distribution pairing number 200.

4.5.1 Security implications

The "distribution-effect" increases bitstring diversity, but is limited to the number of possible μ_{test} and Rng_{test} values (we proposed 10 each earlier). However, the distribution parameters are determined by the *Path-Select-Mask*, which provides an exponential *n*-choose-*k* component to bitstring diversity. Therefore, an adversary who is attempting to clone the behavior of a token, might be able to simulate and catalog all possible combinations of the other user-defined parameters, but he/she will not be able to do this for all possible sets of PN. For authentication applications, the adversary will need to wait for the verifier to send the challenges and *Path-Select-Masks* in order to construct the distribution using simulation data, before he/she can respond with a predicted response bitstring. This adds considerable time and complexity to an impersonation attack, beyond that required to build an accurate model.

4.6 Conclusions

A novel PUF-based entropy-enhancing technique is proposed that is based on biasing distribution data used in bitstring construction using path selection and a linear transformation. The technique changes the response bit values associated with a fixed set of path delays, making the task of model-building more difficult.

Chapter 5

A Novel Offset Method for Improving Bitstring Quality of a Hardware-Embedded Delay PUF

Statistical properties including uniqueness, randomness and reproducibility are commonly used as metrics for Physical Unclonable Functions (PUFs). When PUFs are used in authentication protocols, the first two metrics are critically important to the overall security of the system. Authentication reveals the bitstrings (and helper data if used) to the an adversary, and makes the PUF vulnerable to tactics that can lead to successful cloning and impersonation. In this chapter, we investigate security metrics including Entropy, uniqueness and randomness using hardware data collected from a set of 45 Xilinx Zynq FPGAs which implements HELP. A novel technique is proposed that allows the verifier to randomly or purposefully offset path delays to obfuscate (in the former case) and/or tune (in the latter case) the bitstring generation process. We show that tuning additionally has a significant impact on the statistical quality of the bitstrings.

5.1 Introduction

Security and trust have become critically important for a wide range of existing and emerging microelectronic systems including those embedded in aerospace and defense, industrial ICS and SCADA environments, automotive and autonomous vehicles, data centers, communications and medical healthcare devices. The vulnerability of these systems is increasing with the proliferation of internet-enabled connectivity and unsupervised in-field deployment. Authentication and encryption are heavily used for ensuring data integrity and privacy of communications between communicating devices. These protocols require keys and bitstrings (secrets) to be stored in persistent, non-volatile memory (NVM). Current methods utilizing a “burned-in” key represents a vulnerability, particularly in fielded systems where adversaries can access the hardware and carry out probing and other invasive attacks uninhibited. Physical Unclonable Functions or PUFs on the other hand provide the alternative to NVM key storage, and for the generation of unique and untrackable authentication information.

PUFs extracts random information (entropy) from variations in the physical and electrical properties of ICs, that are unique to each IC, as a means of generating digital secrets (bitstrings). The type and amount of information available to a PUF through these physical-layer variations in the chip are critically important security properties, and are the most often cited benefits of PUFs over conventional NVM-based alternatives.

A PUF is defined by a source of on-chip electrical variations. The hardware-embedded Delay PUF (HELP) analyzed in this paper generates bitstrings from delay variations that occur along paths in an on-chip macro (**functional unit**), such as a cryptographic primitive. Therefore, the circuit structure that HELP utilizes as a source of random information differs from traditional PUF architectures which use precisely placed and routed arrays of identically designed components. In contract, HELP imposes no restrictions on the physical layout characteristics of the entropy source.

This departure from a traditional definition of a PUF architecture provides both advantages and disadvantages. An important advantage is related to the effort involved in constructing the functional unit and the diversity in the number of possible instantiations. For example, commercial logic synthesis tools such as Xilinx Vivado can be used to quickly build multiple different instantiations of the functional unit. Each instantiation is identical in function but has a unique circuit architecture and therefore, a unique set of path delays.

The widely varying nature of path delays in an arbitrarily synthesized functional unit represents the disadvantage. A PUF requires the components, which are being compared to generate a bit, to behave as a random variable, where approx. half of the chips in the population generate a ‘0’ for any arbitrary pair of components while the other half generate a ‘1’. Although it is possible to identify paths that are nearly equal in delay and restrict comparisons within these groups of paths, this greatly constrains the number of candidate path combinations and adds complexity by requiring accurate timing analysis and other operational constraints. Moreover, the analysis needs to repeated for each new instantiation.

HELP addresses the path length bias issue by applying a modulus operation to the measured path delays. The modulus operator computes the remainder after dividing the path delay by specified constant, i.e, the modulus. The modulus is chosen to ideally eliminate the large bias which can be present when paths vary widely in length (and delay), while simultaneously preserving the smaller variations that occur because of random processes, e.g., within-die process variations. The best choice of the modulus makes *any* arbitrary pairings of path delays a random variable.

In order to ensure that bias is removed for every path pairing combination, the modulus needs to be as small as possible. This is true because the magnitude of the randomly varying component of path delays differs based on the length of the paths used in each pairing. Unfortunately, the modulus is lower bounded by measurement, temperature, supply voltage noise sources. Therefore, the range of *suitable* moduli that achieve the PUF's primary goals of producing unique, random and reproducible bitstrings is limited.

In this chapter, we propose an offset technique that can be used to increase the range of suitable moduli, that builds on the concept first presented in [66], Section 5.3. We describe the technique in reference to a PUF-based authentication scenario, which occurs between a hardware token and a verifier. In our proposed authentication protocol, a set of path delays are collected and stored in a secure database during the enrollment process, i.e., before the token is released for field use. The verifier also computes and stores the *median* values of each path delay using the enrollment data of the tokens. During authentication, the verifier selects a modulus and then computes the difference between the median path delay and the modulus, and encodes the differences (called offsets) in the challenge data sent to the token. The token and verifier add the offsets to the measured (stored for verifier) path delays before computing the corresponding bit. An offset effectively shifts the distribution of a given path delay such that approx. half of the chips generate a '0' and half generate a '1'.

We evaluate the effectiveness of the offset technique using inter-chip hamming distance and Entropy metrics from data collected from a set of Xilinx Zynq FPGAs. The results are compared with those obtained using the original 'unshifted' data. The remainder of this chapter is organized as follows. Section 5.2 presents related work and Section 5.3 provides an overview of the HELP PUF. Section 5.4 presents experimental results and Conclusions are presented in Section 5.5.

5.2 Related Work

References [37] and [66] describe previous research on HELP. However, the offset method and analysis presented in this paper are new contributions. To our knowledge, the offset method has not been proposed elsewhere and is a technique that is only applicable to PUFs that utilize a modulus method to remove undesirable bias effects as is done in HELP.

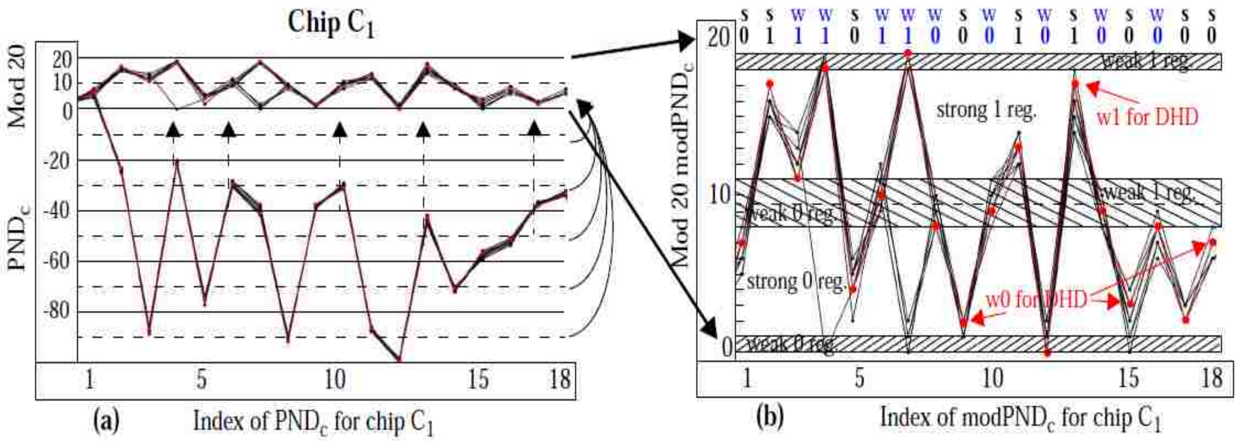


Fig. 5.1 (a) Conversion from PND_c to $modPND_c$ and (b) Strong/Weak PND_c classification using margining.

5.3 HELP Overview

The instantiation of the HELP processing engine and the PN processing procedures are described in Chapter 4, Section 4.3. Note that after the TVCOMP procedure, the variations that remain in the PND_c are those introduced by within-die variations (**WDV**) and *uncompensated* TV noise (**UC-TVNoise**). UC-TVNoise sets the low bound on the range of suitable moduli as discussed earlier, while WDV defines the upper bound. The offset method described below is designed to extend the range of suitable moduli upwards while maintaining or improving the randomness and uniqueness statistical quality metrics in the generated bitstrings.

5.3.1 Bitstring Generation

The bitstring generation process uses a fifth user-specifiied parameter, called the *Margin*, as a means of improving the reliability of the bitstring regeneration process. The bottom portion of Fig. 5.1(a) plots 18 of the 2048 PND_c from $Chip_1$ along the x-axis. The red curve line-connects the data points obtained under **enrollment** conditions (25^{o}C, 1.00V) while the black curves line-connects data points under a set of **regeneration** TV corners, which in our experiments, is all combinations of temperatures -40^{o}C, 25^{o}C, 85^{o}C, 100^{o}C with supply voltages 0.95V, 1.00V and 1.05V.

The curves plotted along the top of Fig. 5.1(a) show the $modPND_c$ values after a modulus of 20 is applied. Fig. 5.1(b) enlarges the upper portion of Fig. 5.1(a) and includes a set of margins of size 2 surrounding two strong bit regions of size 6. Designators along the top given as ‘s0’, ‘s1’, ‘w0’ and ‘w1’ classify each of the enrollment data points as either a strong 0 or 1, or a weak 0 or 1, resp. Data points that fall on or within the hatched areas are classified as weak as a mechanism to avoid bit flip errors introduced by UC-TVNoise that occurs during regeneration.

The Margin method improves bitstring reproducibility by eliminating data points classified as ‘weak’ in the bitstring generation process. For example, the data points at indexes 4, 6, 7, 8, 10 and 14 would introduce bit flip errors at one or more of the TV corners during regeneration because at least one of the regeneration data points is in the opposite bit value region from the corresponding enrollment value. We refer to this bitstring generation technique as the **Single Helper Data** (SHD) scheme since the classification of the modPND_c as strong or weak is determined solely by the enrollment data.

A second technique, referred to as the **Dual Helper Data** (DHD) scheme, requires that both the enrollment and regeneration modPND_c be in strong bit regions before allowing the bit to be used in the bitstring during regeneration. The *helper data* for enrollment and regeneration, which represents the classification of the modPND_c as strong or weak, is bitwise ‘AND’ed, and then both the enrollment and regeneration bitstrings are generated (the enrollment data is assumed to be collected earlier in time and stored on a secure server). The DHD scheme doubles the protection provided by the margin against bit flip errors because the modPND_c produced during regeneration must now change and move across both a ‘0’ and ‘1’ margin before it can introduce a bit flip error. This is true because both the enrollment and regeneration modPND_c must be classified as strong to be included in the bitstring and the strong bit regions are separated by 2**Margin*.

Fig. 5.1 highlights four cases where an enrollment-classified strong bit would be reclassified as weak in the DHD scheme because 1 or more of the regeneration modPND_c

falls within a weak region. This shows that in addition to doubling the protection against bit flip errors, the DHD scheme can potentially produce different bitstrings each time the chip regenerates it.

5.4 Offset Method

The offset method is low cost because it leverages the enrollment data already available on the verifier, and encodes offsets into a set of unused bit positions in the existing *Path-Select-Mask*s sent to the token.

As we described in Section 5.3, a ‘challenge’ for HELP is defined as a 2-vector sequence that is applied to the PIs of the functional unit and a *Path-Select-Mask* which is used to select a subset of the measured path delays from those that are available on the POs. The *Path-Select-Mask* increases the response space of HELP exponentially because it allows the verifier to implement an *n*-choose-*k* strategy. For example, assume that a sequence of challenges produces a set of 5,000 rising PN and a set of 5,000 falling PN, from which the server selects a subset of 2048 from each set. The number of ways of choosing 2048 from 5000 is given by Eq. (4.3).

The *Path-Select-Mask* actually encodes three possibilities, 1) a ‘1’ is stored for POs when the paths driving the POs are sensitized (therefore a delay value will be measured and used by HELP in bitstring generation), 2) a ‘0’ is stored when a path is sensitized but is not going to be used by HELP and 3) a ‘0’ is also stored when no transition occurs on the PO. The offset method uses the bit positions for POs that do not have transitions (scenario 3) to store offset information. The token can easily distinguish cases 2) and 3) based on whether or not a path delay was produced (recall that all paths that have transitions are timed simultaneously). A typical sequence of challenges for HELP consists of approx. 400 vectors, assuming 10 of approx. 32 sensitized paths are selected for the bitstring generation process (a total of 4096 PNs are needed). Therefore, the 400 *Path-Select-Mask*s have approx. 400 * 32 = 12,800 unused bit positions available for use by the offset method. The offset method requires either 2048 or 4096 of these bit positions for encoding either a 1-bit or 2-bit offset for each of the 2048 PND_cs.

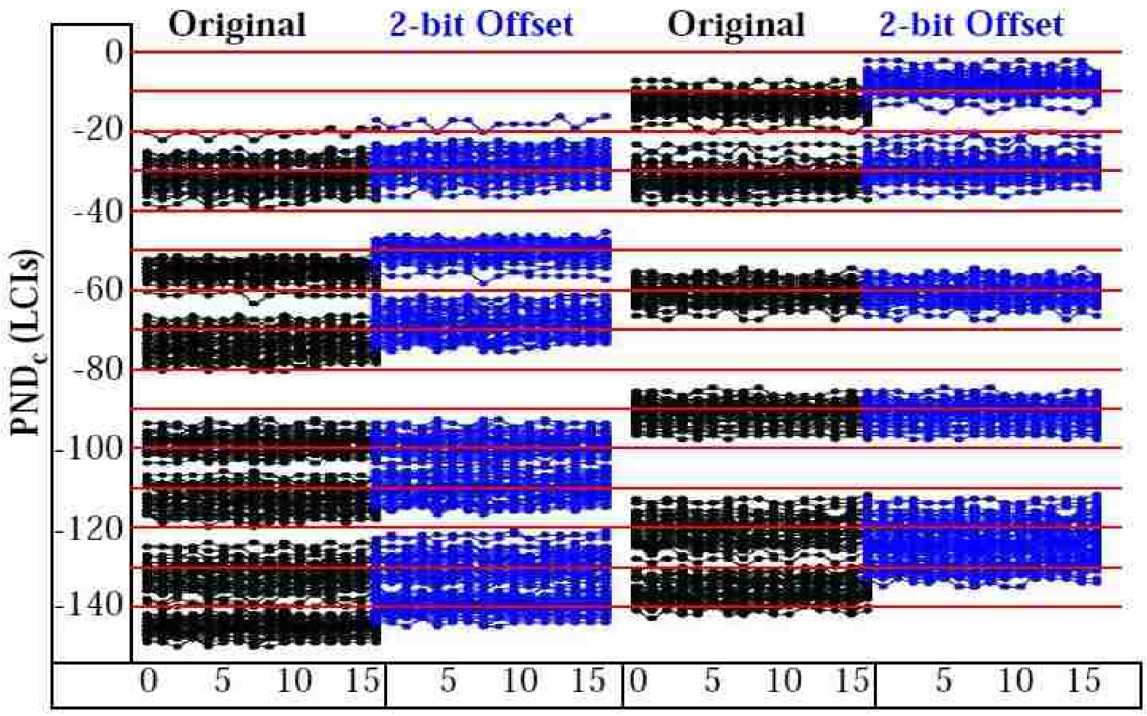


Fig. 5.2 Illustration of the Offset Effect on PNDc

The offset method is designed to shift individual PND_c upwards as a means of centering the population around the one of the 0-1 lines defined by the modulus. This is accomplished after the token and verifier decide on a set of parameters for the authentication round, which includes the *Modulus*. The verifier then selects a set of 2-vector input sequences and randomly chooses 4096 PNs. The *Path-Select-Mask* is updated with the PN selection choices. Before transmitting these challenges to the token, the verifier accesses a set of pre-computed median values for the selected PNs and applies the PNDiff and TVCOMP processes described in Section 5.3 (see Fig. 4.1) to the set of median PNs to obtain a set of modPND$_{c\text{-median}}$ values.

The modPND$_{c\text{-median}}$ values can then be used to compute a set of offsets that will be used to shift the measured and stored PND$_c$ on the token and verifier respectively. The optimal offset values for each PND$_c$ are the absolute value of (*Modulus/2* - modPN$_{c\text{-median}}$). However, encoding the optimal value requires up to log$_2$(*Modulus*/2) bits, e.g, for a *Modulus* of 30, we need 4 bits to encode the offsets. Our analysis presented below shows that most of the benefit of the offset method is obtained with 1-bit and 2-bit encoding schemes. For example, a 1-bit encoding either indicates that the PND$_c$ is to be left as is or shifted by 1/4**Modulus*. A 2-bit scheme allows shifts of 1/8, 2/8 and 3/8**Modulus*.

Once the offsets are determined, the verifier inserts the encoding for the 2048 PND$_c$ into the first 2048 or 4096 un-used bit positions of the *Path-Select-Mask*. The challenges (2 vector sequences and *Path-Select-Mask*s) are then transmitted to the token. The token applies the challenges to generate the 4096 PNs. The verifier reads-out the PNs for the token from its stored enrollment data and both the token and verifier carry out PNDiff and TVCOMP processes. The appropriate offsets are then added into the individual PND$_c$, and the modPND$_c$ computed to enable the completion of the bitstring generation and subsequent authentication processes.

Fig. 5.2 provides an illustration of the 2-bit offset scheme using a set of 13 PND_c both before the offset is added (labeled ‘Original’) and after (labeled ‘2-bit Offset’) in two separate columns for better clarity. The line-connected curves represent the PND_c data for each of the 45 chips, with the x-axis representing the 16 TV corners referenced earlier. The offset shifts the entire distribution upwards towards one of the 0-1 lines given as 0 and 10 for the *Modulus* of 20 used in this analysis. Although not optimally placed, the shifted ‘2-bit Offset’ data shown on the immediate right of the ‘Original’ data show the trend of the data sets to center of the 0-1 lines drawn in red. This simple transformation significantly improves bitstring statistical metrics as discussed in the next section.

5.5 Experimental Results

We applied the offset method to the data collected from a set of 45 Xilinx Zynq FPGAs. The results are shown as a series of bar graphs in Fig. 5.3 to make it easy to see the effect of the offset method. The first 3 rows show results using a Mean scaling factor for μ_{ref} and Rng_{ref}, computed as the mean value of the μ_{test} and Rng_{test} from all PND distributions across all chips and TV corners. Rows 4, 5 and 6 show the results using the maximum (Max.) scaling factor associated with one of the chip-TV corner data sets which produces the largest values for μ_{test} and Rng_{test}. Rows 1 and 4 depict the statistical results for the base case in which no offset is used, while rows

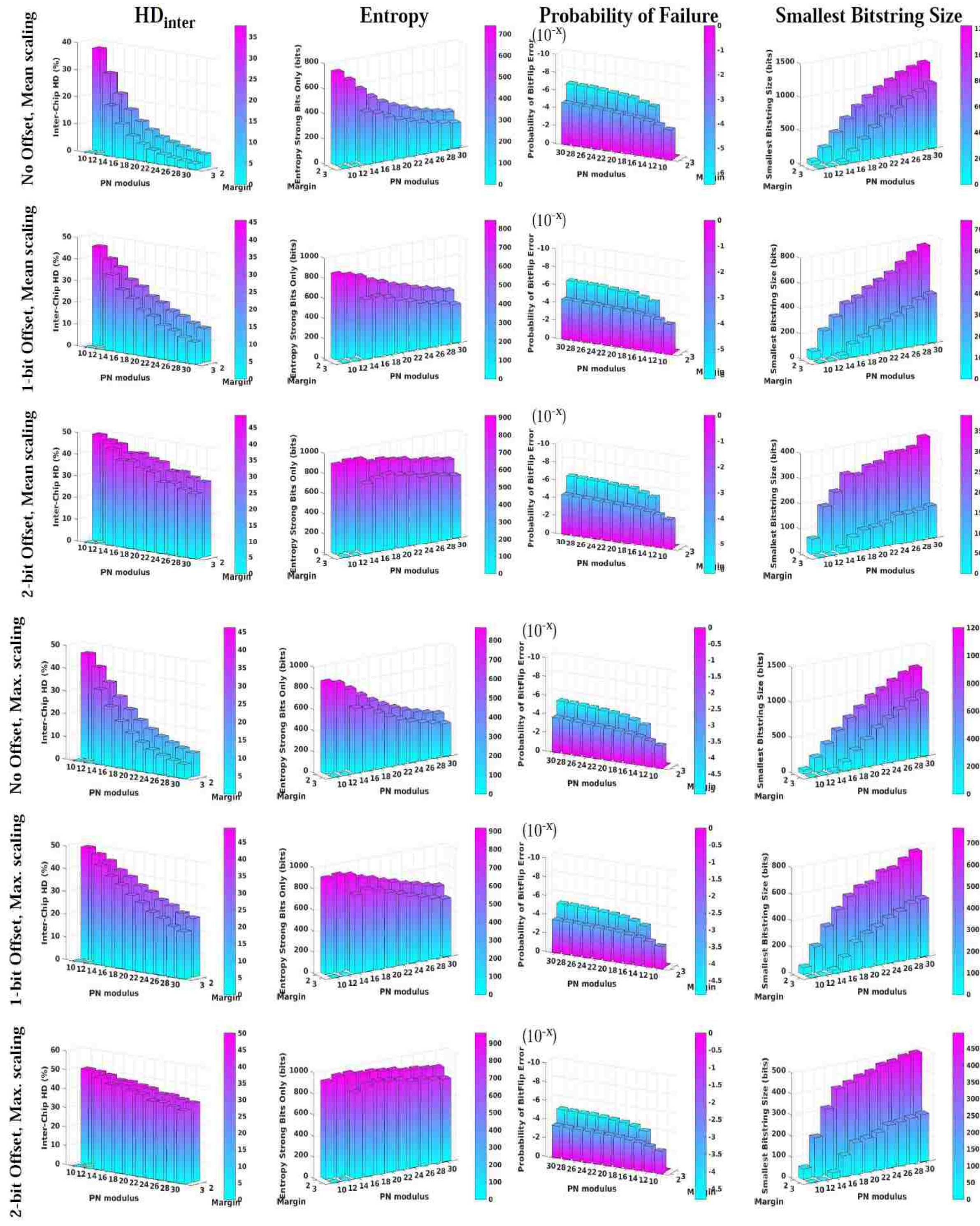


Fig. 5.3 Inter-chip HD, Entropy using strong bits only, Probability of Failure and smallest

bitstring size bar graphs for Margins 2 and 3 and Moduli 10 through 30 within each figure. Rows 1 and 4 show using the Original data using Mean and Maximum μ_{ref} and Rng_{ref} values obtained from the native (before TVCOMP) distributions of the 45 chips. Rows 2 and 5 show the same set of results using a 1-bit offset while rows 3 and 6 give the results using a 2-bit offset. The increasing trend in each of the two groups of 3 rows for Inter-chip HD and Entropy columns show the benefit of the offset method. Column 3 shows the offset method has no impact on reliability. Column 4 illustrates that the smallest bitstring size gets smaller for the offset method because more of the modPND$_c$ are located in the weak bit regions.

2 and 5 show the results using a 1-bit offset and rows 3 and 6 show the results using a 2-bit offset.

Each bar graph portrays the results for one of the statistical metrics, labeled as HD$_{inter}$, Entropy, Probability of Failure and Smallest Bitstring Size in the column headers. The analysis was carried out using Margins 2 and 3 and for Moduli between 10 and 30, as given by the x- and y-axis labels in the graphs. Inter-chip hamming distance (HD$_{inter}$) is computed as the average value across all possible pairing of the enrollment-generated bitstrings from the 45 chips (45*44/2 = 990 pairings) and across a sequence of 256 different pairs of 11-bit *LFSR seeds*. For each pairing, the hamming distance is computed by counting the differences between corresponding bits in the bitstrings of length 2048, but using only bits classified as strong in both bitstrings. The differences are converted to percentages and the mean of the percentages are plotted. Entropy is computed using Eq. (5.1) on the strong bits from each bitstring. The frequency p_i of ‘1’s is computed as the fraction of ‘1’s at each bit position for only those chips of the 45 which identify the bit as strong in the enrollment data.

$$H(X) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i) \qquad (5.1)$$

The Probability of Failure is reported as an exponent *x* from 10^{-x} with a value of -6 indicating 1 chance in 1 million. The HD_{intra} is first computed by pairing the enrollment bitstrings for each chip against each of the 15 regeneration bitstrings. The bits considered are those that remain strong under the DHD scheme described in Section 5.3.1. The average HD_{intra} is computed from the average across the 256 *LFSR seeds*, which is then converted into a probability of failure. The smallest bitstring size is the length of the smallest bitstring produced for a chip under the DHD scheme.

The offset method increases the HD_{intra} and Entropy significantly as illustrated by comparing the bar heights across consecutive columns, while having a near zero effect on the Probability of Failure. However, the offset method reduces the size of the smallest bitstring, particularly for Mean scaling data because more of the modPNDc are located in the weak bit regions. The size of the bitstring can be easily increased by increasing the number of PNs processed beyond 4096 with only a small impact on time and area overhead.

5.6 Security implications

The offset method deals effectively with the loss of entropy-per-bit as the *Modulus* is increased. It is nearly free of cost because it utilizes enrollment information that is already available on the verifier and unused *Path-Select-Mask* bits. The median PND_c

need to be computed for each authentication request but the process is very fast because the PN_{median} can be pre-computed in advance of any authentication requests and the remainder of the process involves only creating the PND_{median} and $PND_{c\text{-}median}$ after the set of PNs are selected. The insertion of the offsets into the *Path-Select-Mask* and the retrieval by the token has negligible time and area cost. The verifier and token add in the offsets before the $modPND_c$ are created, which involves a extra addition operation for each of the 2048 PND_c.

It should be noted that the number of combinations possible using the *Path-Select-Mask*, according to Eq. (4.3), makes the chances that an identical set of challenges will be used more than once for **any authentication to any token** near 0. Therefore, the non-ideal trending of the HD_{inter} and Entropy of the Original scheme for larger moduli is moot. However, the offset scheme provides improved security properties for this rare case and will benefit high security applications. Also note that randomly assigning offset values is also a valid strategy. In this case, the statistical properties of the bitstrings will remain unchanged over the Original scheme but the response space will be obfuscated even for cases in which all other user-defined parameters are help constant.

5.7 Conclusions

An offset method is described in this paper that ‘tunes’ the individual distributions associated with the path delay values. The tuning is designed to center the populations over the 0-1 lines used during the bitstring generation process, as a means of increasing the entropy per bit toward the ideal value of 50%. The offset is very low in overhead, leveraging enrollment information stored by the verifier and integrating offset values into unused challenge bits in the *Path-Select-Mask* component. The technique is demonstrated using data collected from a set of FPGAs to significantly improve the uniqueness and randomness properties of the generated bitstrings.

Chapter 6

Delay Model for the HELP PUF

This chapter proposes a delay model for the HELP PUF and analyzes the feasibility of applying model-building attack on the HELP PUF. In the proposed delay model, the delay values of all sensitized paths under a given challenge vector pair can be represented by the segment delays and the values of the given vector pair. We find that the size of the delay model will increase exponentially with the path length. Also, there exist uncertainty that which input transition dominates the timing of the gate under the circumstance where more than one inputs of the gate possess transitions (no hazard). Although the uncertainty can be reduced by attaching a nominal delay field to each delay segment in the model, the within-die variations that exist across chips make it very difficult to eliminate such uncertainty.

6.1 Delay models for Arbiter PUFs

The basic idea of the classic Arbiter PUF (APUF) is to use the delay difference of two symmetrically designed paths to generate a binary response bit. The APUF consists of k-stage switches each of which possesses four delay segments, i.e., $\Psi_{i,1}$, $\Psi_{i,2}$, $\Psi_{i,3}$ and $\Psi_{i,4}$, for the *i-th* stage, as shown in Fig. 6.1. A transition is launched from the "en" signal and propagates via two separate paths through the k-stage switches. In the *i-th* stage, only one pair of delay segments will be sensitized depending on the *i-th* challenge bit value: the straight segment pair ($\Psi_{i,1}$, $\Psi_{i,2}$) when c[i] = 0 or the crossed segment pair ($\Psi_{i,3}$, $\Psi_{i,4}$) when c[i] = 1. A response bit is generated according to the total delay difference of the two sensitized paths using an arbiter.

An additive linear model is used to describe the mechanism of APUFs. In the model, the total delay difference of the two sensitized paths can be represented by the accumulated delay differences of each stage [1]. The delay difference at the i-th stage is represented by $\beta_i^{c[i]}$, where the superscript c[i] is the i-th

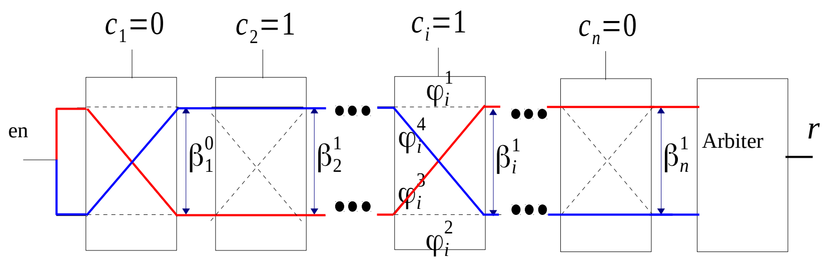


Fig. 6.1 Structure of classic Arbiter PUFs

challenge bit denoting that either the straight segment pair is sensitized (when c[i] = 0) or the crossed pair (when c[i] = 1). Therefore, we have $\beta_i^0 = \Psi_{i,1} - \Psi_{i,2}$ and $\beta_i^1 = \Psi_{i,4} - \Psi_{i,3}$, respectively. We define two (k+1)-dimension vectors $\vec{w}$ and $\vec{\varphi}(\vec{C})$ so that the total delay difference can be represented as: $\Delta t = \vec{w}^T \cdot \vec{\varphi}$. $\vec{w}$ is the parameter vector that encodes segment delays in the APUF stages and the feature vector $\vec{\varphi}(\vec{C})$ is a function of the applied k-bit challenge $\vec{C}$. In more detail,

$$\vec{w} = (w^1, w^2, \ldots, w^k, w^{k+1}) \tag{6.1}$$

where $w^1 = \frac{\beta_1^0 - \beta_1^1}{2}$, $w^i = \frac{\beta_{i-1}^0 + \beta_{i-1}^1 + \beta_i^0 - \beta_i^1}{2}$ and

$$\vec{\varphi}(\vec{C}) = (\varphi'^1(\vec{C}), \varphi'^2(\vec{C}), \ldots, \varphi'^k(\vec{C}), 1) \tag{6.2}$$

where $\vec{\varphi}^{l}(\vec{C}) = \prod_{i=l}^{k} (1-2c[i])$.

The output value of the arbiter is determined by the sign of the total delay difference, i.e., a '0' is generated if Δt < 0 and otherwise a '1' is generated. If we map the output space from {0, 1} to {-1, 1}, then the output r can be represented as:

$$r = f_{PUF} = sign(\vec{w}^{T} \cdot \vec{\varphi}) \qquad (6.3)$$

Equation (5) denotes a linear threshold function (LTF) where $\vec{w}^{T} \cdot \vec{\varphi} = 0$ defines a hyperplane that separates the space of the feature vector $\vec{\varphi}(\vec{C})$ into two half-spaces:

S1 = { $\vec{\varphi}(\vec{C}) \in \{0,1\}^{k+1}$ | $\vec{w}^{T} \cdot \vec{\varphi}(\vec{C}) < 0$ } and

S2 = { $\vec{\varphi}(\vec{C}) \in \{0,1\}^{k+1}$ | $\vec{w}^{T} \cdot \vec{\varphi}(\vec{C}) > 0$ }.

If the feature vector of a give challenge is located in S1, then a '-1' is generated at the output otherwise a “1”. Determining the located space region of the feature vector enables the prediction of the response of the APUF under a given challenge.

6.2 HELP PUF Structure and Working Mechanism

6.2.1 Measuring path delays using clock strobing

The delays of a set of paths are measured by applying a series of launch-capture clocking events (called clock strobing) using Clk1 and Clk2 as shown on the left side of Fig. 2(a). A 2-vector sequence (V1, V2) is applied at the k-bit primary inputs, labeled PI, using the Launch Row FFs as a means of generating logic transitions in the functional unit. The first vector V1 represents the initialization vector. The application of the second vector V2 generates a set of transitions which could be timed by the clock strobing technique. For each repeated application of this 2-vector test sequence, the phase shift between Clk1 and Clk2 is increased by a small fixed Δt. The phase shift value between

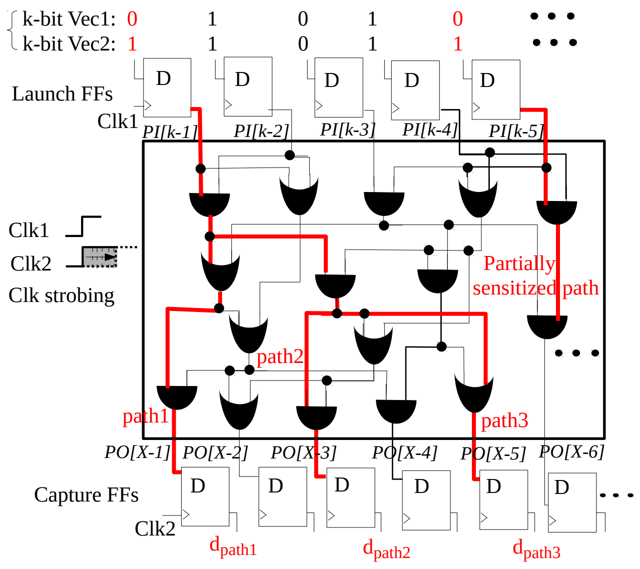
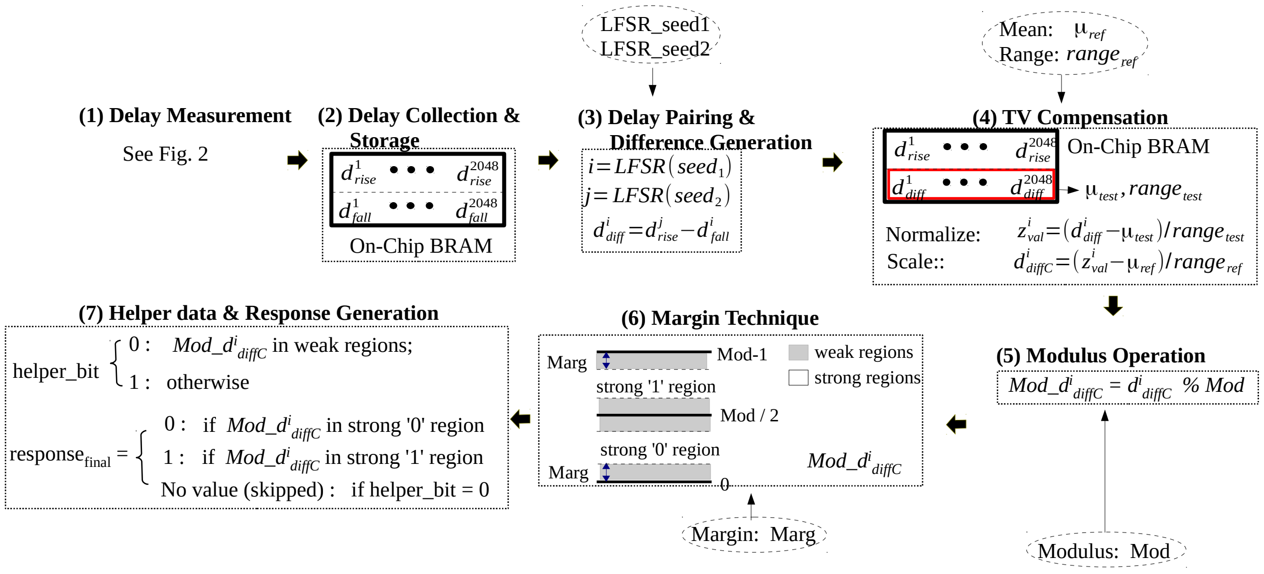


Fig. 6.2 Configuration of the functional unit (FU) and clock strobing method for measuring path delays for HELP PUF.

the two clocks is digitally controlled, and is referred to as the launch-capture interval (LCI). The smallest LCI that allows the propagating edge along a path starting from a Launch FF to be captured in a capture FF is used as the digitized delay value for the path. The digital delay values for a large number of paths can be obtained by repeating the clock strobing operation (with a gradually increasing LCI) for multiple 2-vector test sequences.

6.2.2 Delay processing

The sensitized path delays will be stored into an on-chip BRAM for processing to generate the final response bitstring. A flow of the response bit generation process is depicted in Fig 2(b) and described as the procedures as below:



(1) Delay Measurement
See Fig. 2
(2) Delay Collection & Storage
On-Chip BRAM
LFSR_seed1
LFSR_seed2
(3) Delay Pairing & Difference Generation
$i=LFSR(seed_1)$
$j=LFSR(seed_2)$
$d^i_{diff}=d^j_{rise}-d^i_{fall}$
Mean: μ_{ref}
Range: $range_{ref}$
(4) TV Compensation
On-Chip BRAM
$\mu_{test}, range_{test}$
Normalize: $z^i_{val}=(d^i_{diff}-\mu_{test})/range_{test}$
Scale:: $d^i_{diffC}=(z^i_{val}-\mu_{ref})/range_{ref}$
(5) Modulus Operation
$Mod\_d^i_{diffC} = d^i_{diffC}$ % Mod
Modulus: Mod
(6) Margin Technique
Marg
Mod-1
strong '1' region
Mod / 2
strong '0' region
0
weak regions
strong regions
$Mod\_d^i_{diffC}$
Margin: Marg
(7) Helper data & Response Generation
helper_bit 0 : $Mod\_d^i_{diffC}$ in weak regions; 1 : otherwise
response$_{final}$ = 0 : if $Mod\_d^i_{diffC}$ in strong '0' region; 1 : if $Mod\_d^i_{diffC}$ in strong '1' region; No value (skipped) : if helper_bit = 0

Fig. 6.3 Response bit generation flow and input parameters for HELP PUF

1. Delay measurement as shown in Fig. 6.2.

2. Delay collection and storage: A on-chip BRAM is used to store the collected 4096 path delay values, half of which are sensitized under rising vector pairs and other half using falling vectors. They are denoted as d^j_{rise} and d^i_{fall} respectively.

3. Delay pairing and difference generation: One rising delay and one falling delay are selected to construct a delay pair using two 11–bit LFSRs. The two 11-bit LFSR seed values are the 1st user-defined parameter. A delay difference value is generated by the paired delay values as $d^i_{diff}=d^j_{rise}-d^i_{fall}$.

4. TV compensation: The delay difference is normalized by using the mean 'u_{test}' and range 'rng_{test}' derived from the distribution of the 2048 d^{i}_{diff} values and then rescaled to be the compensated delay value of d^{i}_{diffC}. The 2nd and 3rd user-defined parameters u_{ref} and rng_{ref} are used for the rescaling as shown in the procedure (4) of Fig. 6.3.

5. Modulus operation: A modulus operation is applied to the 2048 d^{i}_{diffC} values using a modulus value Mod used as the 4th user-defined parameter to get the $Mod\_d^{i}_{diffC}$ values.

6. Margin Technique: a 5th user-defined parameter value called margin is provided as a way of eliminating those $Mod\_d^{i}_{diffC}$ which are close to the '0-1' boundary lines to cause bit flips.

7. Helper data and strong bitstring generation: only those $Mod\_d^{i}_{diffC}$ that locate within the strong regions generate a response bit with the helper data bit assigned as 1. A zero value is assigned to the helper data bit for those $Mod\_d^{i}_{diffC}$ that locate within weak regions.

6.3 Sensitizing delay segments for HELP PUF

6.3.1 Differences between APUF and HELP PUF

In this section, we will discuss the structural similarities and differences between APUF and HELP PUF. As mentioned above, HELP measures path delays of the functional unit and generates a response bit '0' or '1' according to the location of the

$Mod\_d_{diffC}^{i}$ value that is derived from a pair of path delays.

Arbiter PUFs can be generalized as an additive linear model since the total delay difference of the two sensitized paths is the sum of the delay differences of each individual stage. Although HELP PUF also utilizes the delay difference of two sensitized paths for response generation, there exist several critical differences that make it impossible to use a linear delay model for HELP:

1. APUF leverages delay difference of two k-stage paths that are identically designed. For each stage, there are only two fixed pairs of delay segments to select: either the straight pair or the crossed pair. Which pair is selected as the *i-th* stage participated segments of the two sensitized paths only depends on the *i-th* bit challenge value. Such systematically stage-wised structure across all k stages makes APUF fits into a simple linear additive model. In the model, the delay difference of each individual stage can be represented by a function of one-bit challenge value. HELP PUF, on the other hand, can never be generalized by such linear additive model since much more complicated 'challenge-to-sensitized-segment' relationship exist for the two sensitized paths. First, the two paths are with different stage lengths and are separately sensitized by two independent challenge pairs (one rising and one falling), no one-stage-to-one-challenge-bit relationship exists. Second, the number of possibly paired delay segments at each stage (with the same depth) is in the order of n^2, where n is the number of structural paths of the functional unit. Third, whether a delay segment of the *i-th* stage gets sensitized depends on multiple challenge bits that are present in the boolean expression of this delay segment.

2. A single challenge pattern is always guaranteed to sensitize only two paths for APUFs. However, a given challenge pair for HELP could possibly sensitize any number of structural paths ranges from 0 to the number of primary outputs. In fact, it equals to the number of primary outputs that have transitions. For instance, 3 structural paths get sensitized under the given challenge pair in Fig. 6.2.

3. Applying a single challenge pattern to APUF is guaranteed to generate one corresponding response bit on the fly. However, this is not true for HELP in three aspects: First, there exists no corresponding response value for those applied challenge pairs that sensitize zero structural path. Second, the response generation process will not even get started until 2048 rising and 2048 falling delays have been collected in the on-chip BRAM. Third, which pair of rising delay and falling delay will be selected from the two 2048 sets further depends on the values of the two LFSRs (procedure (3) in Fig. 6.3).

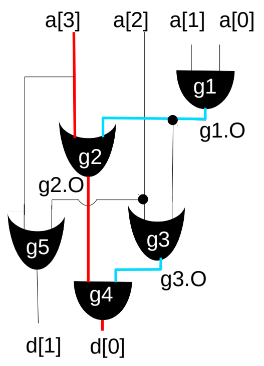


Fig. 6.4 Sample circuit for illustration of the delay model of HELP

6.3.2 Preliminaries of delay segments and path sensitization for HELP

For illustration purpose, we describe the process of building a delay model for HELP using a simple sample function unit shown in Fig. 6.4. The sample functional unit consists of five two-input logic gates g1~g5 (AND and OR) and has four primary inputs a[3]~a[0] and two primary output d[1] and d[0].

The HELP PUF leverages the delay of sensitized structural paths as the source of entropy. A structural path starts from a primary input and ends at a primary output of the functional unit. To achieve glitch-free operation, the functional unit of HELP is implemented using positive logic gates [66] and the applied two-vector sequence either possesses '0' to '1' transitions or '1' to '0' transitions but not both. Such 'glitch-free' operation maintains the transition direction initialized at the primary input unchanged along the sensitized structural path. For each structural path, there are two corresponding transition paths: the rising path and falling path. The rising (falling) path is traversed by a rising (falling) transition initialized in the primary input and propagating through logic gates along the path. We regard the two transition paths as two distinct paths in the following discussions. The nodes along a sensitized path is called the on-path nodes, and the gates along the sensitized path have only one on-path input and all the others being off-path inputs. For a sensitized gate at the *i-th* stage of a sensitized path, two delay elements contribute to the path delay: 1) the interconnect delay that a signal transition travels from the output of a previous gate (through a fanout branch) to the on-path input of the destination gate; 2) the switching gate delay associated with a transition propagates from the on-path input to the gate output. For convenience, the combination of these two delay elements at each stage is defined as a delay segment so that a path delay can be represented by the sum of these segment delays at each stage. Similar to the transition paths, there are two 'transition' delay segments associated with one physical delay segment: the rising and the falling delay segments.

The concept of delay segments is introduced for the convenience of representing a path delay using individual delay units. A delay segment can be regarded as an input segment of a gate, which are considered separate for each input of the gate. Therefore, we denote a delay segment using the corresponding gate input node and the gate output node in the form of (gateInput, gateOutput). For example, there are two delay segments associated with gate g2 in Fig. 6.4 denoted as (a[3], g2.O) and (g1.O, g2.O), respectively, where 'gX.O' represents the output node of gate 'gX'.

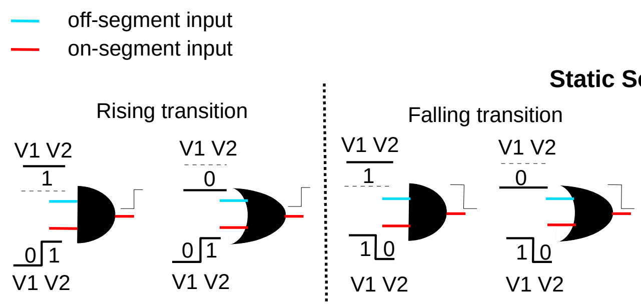
The condition of sensitizing a transition path is that the applied two-vector sequence will initialize a transition at the primary input and the transition will propagate through every delay segment along the path. This is equivalent to the requirement that the conditions of sensitizing each delay segment along the path are met simultaneously, as is discussed in the following section.

6.3.3 Condition of sensitizing a delay segment

Similar to the 'on-path' input of a gate, we call the input that involves the delay segment as the 'on-segment' input and all the rest as 'off-segment' inputs of a gate. The condition of sensitizing an input segment of a gate depends on the logic function of the gate and the off-segment inputs values. For simplicity, we analyze the condition of sensitizing a delay segment of the 2-input AND and the 2-input OR gates for the rising and falling transitions as below. Note that the controlling and non-controlling input value for an AND gate are '0' and '1', respectively, and vice versa for an OR gate.

There are two cases where a delay segment of a gate gets sensitized and dominates the timing of the gate. The first case is called 'static sensitization' where all the off-segment inputs of the gate hold a constant non-controlling value and only the on-segment

Static Sensitization

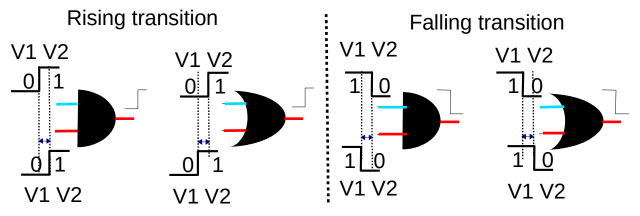


(a) Static sensitization for 2-input AND and OR gates under rising and falling transition.

Gate type	Off-segment inputs: In_{off}		On-segment inputs: In_{on}	
	V1, V2	Boolean Expr.	V1, V2	Boolean Expr.
AND	(V1,V2) = (1,1)	$(\mathrm{In}_{off})_{v1} \& (\mathrm{In}_{off})_{v2}$	(V1,V2)=(0,1)	$\overline{(\mathrm{In}_{on})_{v1}} \& (\mathrm{In}_{on})_{v2}$
OR	(V1,V2)= (0,0)	$\overline{(\mathrm{In}_{off})_{v1}} \& \overline{(\mathrm{In}_{off})_{v2}}$	(V1,V2)=(0,1)	$\overline{(\mathrm{In}_{on})_{v1}} \& (\mathrm{In}_{on})_{v2}$

(b) Boolean expressions that represent conditions of static sensitization under rising transition.

Dynamic Sensitization



(c) Dynamic sensitization for 2-input AND and OR gates under rising and falling transition

Gate type	Off-segment inputs: In_{off}		On-segment inputs: In_{on}	
	V1, V2	Boolean Expr.	V1, V2	Boolean Expr.
AND	(V1,V2)=(X,1)	$(\mathrm{In}_{off})_{v2}$	(V1,V2)=(0,1)	$\overline{(\mathrm{In}_{on})_{v1}} \& (\mathrm{In}_{on})_{v2}$
OR	(V1,V2)=(0,X)	$\overline{(\mathrm{In}_{off})_{v1}}$	(V1,V2)=(0,1)	$\overline{(\mathrm{In}_{on})_{v1}} \& (\mathrm{In}_{on})_{v2}$

(d) Boolean expressions that represent conditions of static or dynamic sensitization under rising transition.

Fig. 6.5 Condition of sensitizing a delay segment of the 2-input AND and 2-input OR gate. (a) static sensitization (b) dynamic sensitization

input has a transition, as is illustrated on the left side of Fig. 6.5 (a). Static sensitization is deterministic because the only transition at the on-segment input is guaranteed to cause a transition at the gate output and thus dominates the timing of the gate. In the second case, however, the on-segment input is not the only input that possesses a transition but at least one of the off-segment inputs of the gate, as shown in the graph in Fig. 6.5 (c). We call this case 'dynamic sensitization' because more than more inputs of the gate possess transitions and the input transition that dominates the timing is the one that causes the gate output to change. If the input transitions change from a non-controlling value to a controlling value (1-to-0 for AND gate and 0-to-1 for OR gate), the input transition that occurs first is the one that causes the transition at the gate output and thus dominates the timing. In the opposite case, the input transition that happens last dominates the timing. Fig. 6.5 (c) enumerates all these scenarios where the transition at the on-segment input dominates the timing. Note that in order to guarantee the desired order of input transitions described above, we need the delay information of each on-path segments on previous stages. Therefore, the dynamic sensitization is non-deterministic in the sense that which input transitions happens first depends on the gate delays of previous stages.

In order to represent the conditions discussed above in a delay model, we need to translate these sensitization conditions into corresponding boolean expressions. The final boolean expression is the ANDed result of two sub-expressions: one for the on-segment input and the other for the off-segment inputs of the target gate. The table in Fig. 6.5 (b) lists the boolean expressions that represent the conditions, for the rising transition, where the sensitized target delay segment dominates the timing under static sensitization. The corresponding boolean expressions for the falling transition case can be easily inferred from the falling scenario depicted in Fig. 6.5 (a).

The table in Fig. 6.5 (d) gives the boolean expressions that represent the condition of a general case where either static or dynamic rising sensitization occurs. Note that such general form of boolean expressions is only able to precisely represent the conditions for the static sensitization case because the uncertainty feature of the dynamic sensitization requires gate delay information of previous stages. In order to eliminate the uncertainty of dynamic sensitization, we can attach a delay field to each delay segment that can be derived by running post-timing simulations on the functional unit netlist. This delay field can be referred to latter in the model to decide which input transition dominates the timing under dynamic sensitization scenarios. However, this strategy will be ineffective if the nominal delay difference of two input transitions are so small as in the range of within-die variations (discussed in Section 6.5).

6.3.4 Condition of sensitizing a structural path

With the boolean expressions available that represent the condition of sensitizing each individual delay segment in the functional unit, the conditions of sensitizing a specific path can be represented by the ANDed result of these expressions of every on-path delay segment as below:

$$Expr_{path\_i} = \prod_{j=0}^{k_{path\_i}} Expr_{seg\_j}^{path\_i} \qquad (6.4)$$

where k_{path_i} represents the lengths (number of delay segments) of *path_i* and $Expr_{seg\_j}^{path\_i}$ represents the boolean expression of sensitizing the *j-th* delay segment along *path_i*.

Note that the size of the boolean expression $Expr_{seg\_j}^{path\_i}$ increases exponentially with the depth of the segment (stage index) along the path. This is true since the number of primary input literals in the boolean expressions of a gate input at stage *j* is approx. 2^j if all the gate are 2-input gates.

6.4 Proposed Delay Model for HELP

We define an m-dimentional parameter vector $\vec{w}$ for HELP where the *i-th* element w_i represents the delay of the *i-th* delay segment within the functional unit (m denotes the total number of delay segments). In order to construct the total delay of a structural path, we need to sum up the delays of every on-path delay segments. An on-path vector $\vec{p}$ is defined for each structural path as an *m-dimensional* vector where the *i-th* element p_i represents whether the *i-th* delay segment is an on-path segment (being a '1') or an off-path segment (being a '0'). With $\vec{w}$ and $\vec{p}_{path\_i}$, we are able to denote the total delay of a path d_{path_i} as:

$$d_{path\_i} = \vec{w} \cdot \vec{p}^{T}_{path\_i} \tag{6.5}$$

An on-path matrix $\vec{P}$ can be constructed by putting all the on-path vectors as individual columns as: $\vec{P} = \left[\vec{p}^{T}_{path\_1}, \vec{p}^{T}_{path\_2}, \cdots, \vec{p}^{T}_{path\_i}, \cdots \vec{p}^{T}_{path\_n}\right]$

where n represents the total number of transition paths in the functional unit. With $\vec{w}$ and $\vec{P}$, we can derive an n-dimensional path delay vector $\vec{d}$ as:

$$\begin{aligned} \vec{d} = \vec{w} \cdot \vec{P} = \vec{w} \cdot \left[\vec{p}^{T}_{path\_1}, \vec{p}^{T}_{path\_2}, \cdots, \vec{p}^{T}_{path\_i}, \cdots, \vec{p}^{T}_{path\_n}\right] \\ = \left[d_{path\_1}, d_{path\_2}, \cdots, d_{path\_i}, \cdots, d_{path\_n}\right] \end{aligned} \tag{6.6}$$

The purpose of the delay model is to represent the sensitized path delays under the applied vector pair. If we use the boolean expressions that represent the conditions of sensitizing each transition paths to construct a $n \times n$ conditional diagonal matrix $\vec{E}$ as:

$$\vec{E} = \begin{bmatrix} Expr_{path\_1} & 0 & 0 & \cdots & A_{1n} \\ 0 & Expr_{path\_2} & 0 & & 0 \\ 0 & 0 & Expr_{path\_3} & & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & Expr_{path\_n} \end{bmatrix} \qquad (6.7)$$

With the conditional diagonal matrix $\vec{E}$ and the path delay vector $\vec{d}$, we are able to get an n-dimensional sensitized_path vector $\vec{SP}$ that represent all the sensitized transition paths under any applied vector pair $\vec{C}_{V1,V2}$ by the equation:

$$\vec{SP} = \vec{d} \cdot \vec{E} = [d_{SP1}, d_{SP2}, \cdots, d_{SPn}] \qquad (6.8)$$

where d_{SPi} represents the sensitized path delay value, being either 0 (if *path_i* is not sensitized) or d_{path_i} (if *path_i* is sensitized under vector pair $\vec{C}_{V1,V2}$).

6.5 Exponential scaling of the Expr$_{path}$ size with path length

Using the delay model proposed in the above sub-section, all the sensitized paths can be represented by individual segment delays along the path under any applied vector pair. However, as discussed in Section 6.3.4, the size of the boolean Expression for the on-path segment increases exponentially with its depth. This feature will result in the fact that the size of the path expression $Expr_{path}$ of a path with length k will be approx. r^k , where r is the average number of inputs per gate along the path. This indicates that constructing the delay model will be infeasible if the max path length increases to a certain value like 30 (2^{30} ~ 1 billion).

6.5 Unsolvable uncertainty introduced by within-die variations

As is mentioned in Section 6.3.3, we can attach a nominal delay field (gained by post-timing simulation) to each segment to reduce the uncertainty that which input transition dominate the timing under the dynamic sensitization. However, if the nominal delay differences of the multiple input transitions are so small that they are in the range of within-die variations, then which input transition dominates the timing of the gate will vary across chip-to-chip. Such with-die variations are random, uncontrollable and unpredictable across chips, therefore the uncertainty is very difficult to be eliminated in the delay model.

Chapter 7

Future Work

In chapter 2, the results of applying the proposed scheme to the resistance distribution data in the public domain show that the bitstring regeneration achieved zero bit-flip without any type of helper data. This needs to be further verified on real physical NVM devices in terms of whether the resistance distribution will skew with increasing number of regenerations. If so, we need to set up a mechanism that periodically re-writes the old resistance state to each NVM cell to guarantee long-term, reliable bitstring regenerations.

The experiments presented in chapter 3 analyzes various types of underlying entropy source within the HELP PUF for both glitchy and glitch-free functional units. What we expect is that the magnitude of the delay variations could be multiple times larger than the measurement and temperature/voltage noise (TV noise). Although we used a temperature/voltage compensation technique (TVCOMP) to reduce the TV noise, there are still a small portion of path delays that can not be compensated properly. Future work includes proposing some screening methods to exclude those paths that possess “uncompensated TV noise”. Since the 'uncompensated TV noise' exhibits systematic behavior, further research could extend to investigate solutions to eliminate such systematic behavior, e.g., schemes like dividing the population into separate groups for TV-compensention.

Chapter 4 demonstrates the difficulty against model-building attack introduced by the 'distribution effect' of HELP PUF. Future work includes applying various machine learning algorithms to demonstrate how difficult it is to build an accurate model of HELP to predict responses of arbitrary challenges.

The results in Chapter 5 show the effectiveness of a novel 'offset method' that deals with the bias issue of HELP. We have demonstrated the improved inter Hamming Distance for the modulus ranges from 10 to 30. Future work may include investigating the effectiveness of applying the offset method to larger modulus values.

Chapter 6 proposes a delay model for the HELP PUF. The complexity of the delay model indicates how difficult it is to launch model-building attack to break the HELP PUF. Future work includes applying different machine-learning algorithms to learn the challenge-response relationship.

Chapter 8

Conclusions

In Chapter 1, I introduced the basic metrics for leveraging qualities of Physically Unclonable Functions and reviewed different schemes of helper data that are targeted at addressing the reliability issue of PUFs.

In Chapter 2, I proposed a new type of PUF structure that is able to eliminate the helper data during the bitstring regeneration process. The PUF is built based on the unique re-programmability feature of non-volatile memory cells and is described in the context of an emerging nano-device, i.e., memristors. The methodology is applicable to any type of non-volatile memories including flash memory.

In chapter 3, a PUF-based authentication protocol based on the HELP PUF is proposed and various types of entropy source of HELP are investigated. Particularly, three types of delay variations are investigated, namely, 1) within-die variations that occur with individual LUT cells, 2) global variations that occurs across all LUTs on the chip and 3) delay variations introduce by static and dynamic logic hazards. Both glitchy and glitch-free functional units are analyzed in terms of the magnitude of the delay variations with respect to measurement and temperature/voltage noise.

In Chapter 4, we demonstrated that the 'distribution effect' within the HELP PUF processing engine significantly improves the resilience against model-building. The processed delay difference value used for bitstring generation is dependent on the other values that participate in the distribution used for the TVCOMP process, which introduces bitstring diversity. Changing the distribution characteristics like mean and range can be achieved by specifying a user-defined parameter call Path-Select-Mask. Our experiments show that the Path-Select-Mask combined with the TVCOMP process introduces additional entropy beyond that available in a fixed number of path delays, which makes model-building more difficult.

In Chapter 5, we proposed a technique that deals with the bias issue for HELP PUF. Purposely setting the values of the offset bits is very effective in improving the base inter-chip Hamming Distance and the entropy without affecting the bit flip error rate. On the other hand, randomly setting the offset bits by the verifier in the authentication scenario enlarge the response space dramatically using a fixed set of path delay values. The overhead of the method is fairly small the offset bits can be assigned to the unused path-select-mask bits for authentication.

In Chapter 6, a delay model for the HELP PUF is proposed so that the sensitized path delays can be represented by the delay segments and the given applied vector pair. We notice that the size of the delay model increases exponentially with the path length. Also within-die variations makes it very difficult to eliminate the uncertainty that which input transition dominates the timing of a gate if more than one inputs possess transitions.

References

[1] B. Gassend, et al., "Controlled Physical Random Functions," *Conference on Computer Security Applications*, 2002.

[2] B. Skoric, et al., “Robust Key Extraction from Physical Uncloneable Functions”, Chapter in *Applied Cryptography and Network Security*, 2005.

[3] R. Maes, et al., “Low-overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs,” *CHES*, 2009, pp. 332–347.

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

[5] Z. Paral, et al., "Reliable and Efficient PUF-based Key Generation using Pattern Matching," *HOST*, 2011, pp. 128-133, Jun. 2011.

[6] J. Delvaux. et al., “Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation”, Cryptology ePrint Archive: Report 2013/566

[7] A. Maiti , J. Casarona , L. McHale and P. Schaumont “A large scale characterization of RO-PUF", Proc. IEEE Int. Symp. 2010 Hardware-Oriented Security and Trust (HOST), pp. 94-99, 2010.

[8] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, “A statistical test suite for random and pseudorandom number generators for cryptographic applications”, National Institute of Standards and Technology (NIST), special publication 800-22, August 2008.

[9] M.-D. Yu, D. M’Ra ̈hi, S. Devadas and I. Verbauwhede: “Security and Reliability

Properties of Syndrome Coding Techniques Used in PUF Key Generation," in Government Microcircuit Applications & Critical Technology Conference, GOMACTech 2013, pp. 1–4, Mar. 2013.

[10] Y. Dodis, L. Reyzin, A. Smith, ‘‘Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data,’’ Eurocrypt, 2004.

[11] R Maes, A. Herrewege, I. Verbauwhede, "PUFKY: A Fully Functional PUF-based Cryptographic Key Generator," Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2012, LNCS vol. 7428, pp. 302-319

[12] M. Yu, S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," IEEE Design and Test of Computers, Special Issue on Verifying Physical Trustworthiness of ICs and Systems, vol. 27, no. 1, pp. 48-65, Jan./Feb. 2010.

[13] M. Hiller, D. Merli, F. Stumpf, "Complementary IBS: Application Specific Error Correction for PUFs," IEEE Int'l Symposium on Hardware-Oriented Security and Trust (HOST), 2012 .

[14] M. Yu, D. M'Raïhi, R. Sowell, S. Devadas, "Lightweight and Secure PUF Key Storage Using Limits of Machine Learning," Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2011, LNCS vol. 6917, pp. 358-373.

[15] G.E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in Design Automation Conference, DAC 2007, pp. 9-14, Jun. 2007 .

[16] J. Ju, et al., "Bit String Analysis of Physical Unclonable Functions based on Resistance Variations in Metals and Transistors", HOST, 2012, pp. 13-20.

[17] Z. Paral, S. Devadas, “Reliable and Efficient PUF- based Key Generation Using Pattern Matching,” IEEE Int’l Symposium on Hardware-Oriented Security and Trust (HOST), 2011.

[18] R. Chakraborty, et al., “A Transmission Gate Physical Unclonable Function and On-Chip Voltage-to-Digital Conversion Technique”, in Proceedings of the 50th Design Automation Conference, DAC '13, 2013, May. 2013.

[19] Jeroen Delvaux , Dawu Gu, Dries Schellekens and Ingrid Verbauwhede, “Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis ”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015.

[20] P. Koeberl, et al., “Memristor PUFs: A New Generation of Memory-based Physically Unclonable Functions”, DATE, 2013, pp. 428-431, Mar. 2013.

[21] G. S. Rose, et al., “Hardware Security Strategies Exploiting Nanoelectronic Circuits”, ASP-DAC, 2013, pp. 368-372, Jan. 2013.

[22] G. S. Rose, et al., “Foundations of Memristor Based PUF Architectures,” NANOARCH, July 2013.

[23] G. S. Rose, et al., “A Write-Time Based Memristive PUF for Hardware Security Applications”, ICCAD, 2013, pp. 830-833, Nov. 2013.

[24] O. Kavehei, et al., “mrPUF: A Memristive Device based Physical Unclonable Function”, CoRR, 2013.

[25] Le Zhang, et al. “Highly reliable memory-based Physical Unclonable Function using Spin-Transfer Torque MRAM”. ISCAS 2014:2169-2172, June 2014.

[26] Le Zhang, Zhi-Hui Kong, Chip-Hong Chang, Alessandro Cabrini, Guido Torelli:

Exploiting Process Variations and Programming Sensitivity of Phase Change Memory for Reconfigurable Physical Unclonable Functions. IEEE Transactions on Information Forensics and Security 9(6): 921-932 (2014)

[27] Elena loana Vatajelu, Giorgio Di Natale, Marco Indaco, Paolo Prinetto, "STT MRAM-Based PUFs " , DATE, 2015, pp. 872-875, Mar., 2015.

[28] Jayita Das, Kevin Scott, Srinath Rajaram, Drew Burgett, Sanjukta Bhanja , "MRAM PUF: A Novel Geometry Based Magnetic PUF With Integrated CMOS ", IEEE Transactions on Nanotechnology , Issue. 99, 2015.

[29] J. J. Yang, et al., "Memristive Devices for Computing," Nature Nanotechnol., vol. 8, pp. 13–24, Jan. 2013.

[30] D. B. Strukov et al, "The Missing Memristor Found", in Nature, volume 453, pages 80–83, 2008.

[31] Y. Ho, et al., "Dynamical Properties and Design Analysis for NonVolatile Memristor Memories", Trans. CAS, 58-I(4):724–736, 2011.

[32] Q. Xia, et.al. "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic", Nano Letters, vol. 9, no. 10, 2009.

[33] K. Kim, et al., "A Functional Hybrid Memristor Crossbar-array/CMOS System for Data Storage and Neuromorphic Applications," Nano Letters, vol. 12, no. 1, pp. 389–395, 2011.

[34] J. Ju, et al., "Stability Analysis of a Physical Unclonable Function based on Metal Resistance Variations", HOST, 2013, pp. 143-150.

[35] Zhang, A. Henessy, and S. Bhunia, "Robust Counterfeit PCB Detection

Exploiting Intrinsic Trace Impedance Variations”, *VLSI Test Symposium*, April 2015.

[36] F. Saqib, M. Areno, J. Aarestad and J. Plusquellic, “An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function”, *IET Computers & Digital Techniques*, Vol. 8, Issue 6, Nov. 2014, pp. 288-299.

[37] J. Aarestad, J. Plusquellic, D. Acharyya, “Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF”, *HOST*, 2013, pp. 151-158.

[38] http://zedboard.org/product/zedboard

[39] J. Delvaux, D. Gu, R. Peeters and I. Verbauwhede, “A Survey on Lightweight Entity Authentication with Strong PUFs”, *Cryptology ePrint Archive: Report 2014/977*.

[40] R. S. Pappu. Physical One-Way Functions. *PhD thesis*, MIT, 2001.

[41] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, “Silicon Physical Random Functions”, *Conference on Computer and Communications Security*, 2002, pp. 148-160.

[42] L. Bolotny and G. Robins, “Physically Unclonable Function-based Security and Privacy in RFID Systems”, *PerCom*, 2007, pp. 211-220.

[43] E. Ozturk, G. Hammouri, and B. Sunar, “Towards Robust Low Cost Authentication for Pervasive Devices”, *PerCom*, 2008, pp. 170-178.

[44] G. Hammouri, E. Ozturk, and B. Sunar, “A Tamper-Proof and Lightweight Authentication Scheme, *Pervasive and Mobile Computing*, 2008, 807-818.

[45] L. Kulseng, Z. Yu, Y. Wei, and Y. Guan, “Lightweight Mutual Authentication and

Ownership Transfer for RFID Systems”, *INFOCOM*, 2010, pp. 251-255.

[46] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, “Enhancing RFID Security and Privacy by Physically Unclonable Functions”, *Information Security and Cryptography*, 2010, pp. 281-305.

[47] S. Katzenbeisser, Unal Kocabas, V. Van Der Leest, A. Sadeghi, G. J. Schrijen, H. Schroder, and C. Wachsmann, “Recyclable PUFs: Logically Recongurable PUFs”, *CHES*, 2011, pp. 374-389.

[48] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, “Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-enabled RFIDs”, *Vol. 7397 of Lecture Notes in Computer Science*, 2012, pp. 374-389.

[49] U. Kocabas, A. Peter, S. Katzenbeisser, and A. Sadeghi, “Converse PUF-Based Authentication” *TRUST,* 2012, pp. 142-158.

[50] Y. S. Lee, T. Y. Kim, and H. J. Lee, “Mutual Authentication Protocol for Enhanced RFID Security and Anticounterfeiting”, *WAINA*, 2012, pp. 558-563.

[51] Y. Jin, W. Xin, H. Sun, and Z. Chen, “PUF-Based RFID Authentication Protocol against Secret Key Leakage”, *Vol. 7235 of Lecture Notes in Computer Science*, 2012, pp. 318-329.

[52] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, “Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching”, *Symposium on Security and Privacy Workshop*, 2012, pp. 33-44.

[53] Y. Xu and Z. He, “Design of a Security Protocol for Low-Cost RFID”, *WiCOM*, 2012, pp. 1-3.

[54] Y. S. Lee, H. J. Lee, and E. Alasaarela, “Mutual Authentication in Wireless Body Sensor Networks Based on Physical Unclonable Function”, *IWCMC*, 2013, pp. 1314-1318.

[55] M.-D. M. Yu, D. M’Rahi, I. Verbauwhede, and S. Devadas, “A Noise Bifurcation Architecture for Linear Additive Physical Functions, *HOST*, pp. 124-129.

[56] S. T. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. F. Wong, “System-of-PUFs: Multilevel Security for Embedded Systems”, *CODES*, pp. 27:1-27:10, 2014.

[57] S. Nikova, V. Rijmen and M. Schlaffer, “Using Normal Bases for Compact Hardware Implementations of the AES S-Box”, *Security and Cryptography for Networks*, *Lect. Notes in C.S.*, Volume 5229, 2008, pp 236-245.

[58] K. Tiri and I. Verbauwhede, “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”, *DATE*, 2004, pp. 246-251.

[59] http://en.wikipedia.org/wiki/Hamming_distance

[60] NIST: Computer Security Division, Statistical Tests, http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html

[61] S. M. Nowick and C. W. O’Donnell, “On the Existence of Hazard-Free Multi-Level Logic”, ASYNC, 2003.

[62] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, “Handbook of Applied Cryptography,” CRC Press, ISBN: 0-8493-8523-7, Oct. 1996, http://cacr.uwaterloo.ca/hac/

[63] S. P. Skorobogatov, "Semi-Invasive Attacks - A New Approach to Hardware Security Analysis," *Technical Report UCAM-CL-TR-630*, 2005.

[64] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, “Silicon Physical Random Functions”, *Computer and Communication Security Conference*, Nov. 2002.

[65] J. Aarestad, J. Plusquellic, D. Acharyya, “Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF,” Symposium on Hardware-Oriented Security and Trust (HOST), 2013, pp. 151-158.

[66] W. Che, F. Saqib, J. Plusquellic, "PUF-Based Authentication", *ICCAD*, Nov, 2015.

[67] R. van den Berg, B. Skoric, and V. van der Leest, “Bias-based modeling and entropy analysis of PUFs,” in TrustED’13, November 04, 2013, Berlin, Germany.

[68] S. Katzenbeisser, U. Kocabas, V. Rozic, A. Sadeghi, I. Verbauwhede, and C. Wachsmann, “PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon”, CHES 2012, pp. 283-301.

[69] https://en.wikipedia.org/wiki/AES

[70] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," *DATE*, 2004, pp. 246-251.